Paul DuBois

MySQL

Traducere de Mihai Mănăstireanu

Teora

Titlu original: **MySQL**

Teora
Calea Moșilor nr. 211, sector 2, București
fax: 01/210.38.28
e-mail: teora@teora.kappa.ro

Teora – Cartea prin poștă
CP 79-30, cod 72450 București, România
tel: 01/252.14.31
e-mail: cpp@teora.kappa.ro

Copertă: Gheorghe Popescu
Tehnoredactare: Cătălin Șleam
Director Editorial: Diana Rotaru

Președinte: Teodor Răducanu

NOT 4897 CAL MY SQL
ISBN 973-20-0246-8

Printed in Romania

Cuprins

Despre autor

În ultimii 15 ani, **Paul DuBois** a fost administrator de rețea și de baze de date la Universitatea Madison a statului Wisconsin. Este autorul volumelor *Software Portability with imake* și *Using csh & tcsh* (O'Reilly & Associates, ISBN 1-56592-226-3 și 1-56592-132-1) și a redactat materiale pentru manualul de referință MySQL.

Despre referenți

David Axmark a lucrat drept consultant software timp de peste 15 ani. Printre numeroasele proiecte la care a activat se numără un sistem ultramodern de cercetare a pieței (CommonLISP+CLOS+MySQLs-ISAM), grafică industrială avansată (în 32k de RAM), precum și un sistem de scriere a ordinelor de funcționare pentru o companie de electricitate. A scris numeroase linii de program în limbajele 6502 și în asamblorul Z80, BASIC, C, CommonLisp, (Bourne)-Shell și Perl. A început să se implice în MySQL la câțiva ani înainte de prima lansare publică a limbajului. David locuiește în Uppsala, Suedia, cu plantele și calculatoarele lui.

Vijay Chaugule lucrează ca administrator principal de sisteme pentru o companie din Bay Area care manipulează o rețea UNIX de mari dimensiuni. Are opt ani de experiență în gestiunea sistemelor și a rețelelor și a utilizat o mulțime de instrumente comerciale și sub licență GNU. Adoră să scrie scripturi în Perl și să compileze și să instaleze programe GNU. În prezent, folosește MySQL la un proiect pentru `induscrafts.com`.

Chad Cunningham este dezvoltator de sisteme pentru departamentul de matematică a Universității de Stat din Ohio, acolo unde se concentrează asupra dezvoltării unor instrumente de învățare bazate pe Web, care să potențeze procesul tradițional de instruire din sala de curs.

Bill Gerrard este vicepreședinte responsabil cu tehnologia al firmei Daze Networks, Inc. din Huntington Beach, California. Daze Networks execută servicii Internet, inclusiv serverele Web virtuale DigitalDaze, care vă oferă un control complet asupra propriului dumneavoastră server MySQL de baze de date, precum și `DNSCentral.com`, care oferă înregistrarea numelor de domenii, parcare, transmitere prin Web (Web forwarding) și reexpedierea mesajelor de e-mail folosind MySQL ca bază de date back-end.

Jijo George John este îndrumător tehnic în cadrul Diviziei de dezvoltare software a grupului de analiză chimică din cadrul companiei Hewlett-Packard. Are experiență în dezvoltarea de programe de sistem și de aplicație pentru instrumente chimice. A lucrat cu programe apropiate instrumentelor sau dispozitivelor, dar și cu aplicații pentru baze de date de mari dimensiuni. În prezent, folosește MySQL la un proiect pentru `induscrafts.com`.

Fred Read își are propriul său serviciu de consultanță, ServaLan Systems Limited, de 14 ani. ServaLan Systems este specializată în producția de sisteme UNIX în domenii ca telecomunicațiile, bazele de date, finanțe și Internet. În prezent, lucrează la un sistem de comutație pentru telecomunicații. În ultimii trei ani, a întreținut și un furnizor de servicii Internet (PremierISP) și o mică afacere de publicitate în Web (PremierAds). Este însurat cu Helen de nouă ani. Cei doi au trei copii (două fete și un băiat) și locuiesc în Anglia.

Dr. Egon Schmid lucrează la Six Offene Systeme GmbH din Leinfelden-Echterdingen, Germania, unde face parte din echipa de dezvoltare care realizează soluții sofisticate de baze de date bazate pe Web. De asemenea, este membru al grupului de realizare a documentației pentru limbajul PHP. A fost co-autor al primei cărți de PHP scrisă în limba germană.

Jani Tolonen este dezvoltator software al sistemului MySQL. Este un student în vârstă de 23 de ani al Universității de Tehnologie din Helsinki, unde se dedică tehnologiei calculatoarelor și programării de produse software în multe domenii. Jani s-a aflat sub îndrumarea lui Michael "Monty" Widenius, creatorul sistemului MySQL, și dorește să se dedice în continuare dezvoltării acestui sistem. Jani și Monty au devenit „din greșeală" foarte buni prieteni: au fost vecini cu peste 10 ani în urmă. Activitatea lui Jani în domeniul MySQL a început la puțin timp după ce Monty a început să creeze MySQL în 1995.

Michael "Monty" Widenius este un finlandez vorbitor de suedeză, care a lucrat cu baze de date din 1978. A fost angajat de TcX (al cărei co-patron a devenit) din 1981 și a scris în totalitate componenta UNIREG (predecesorul lui MySQL) și cea mai mare parte a programului server MySQL. A studiat fizica la Universitatea de Tehnologie din Helsinki, dar în materie de programare este în cea mai mare parte un autodidact. A lucrat mai ales în Basic, Assembler, C, LISP, Perl, SQL, C++ (în această ordine), dar dorește mereu să examineze limbaje noi, atunci când dispune de timpul necesar. Este însurat și are doi copii (un băiat și o fată – Max și My), două broaște țestoase, un câine și doi șerpi. În prezent, își petrece majoritatea timpului răspunzând la mesaje de e-mail.

Editura New Riders dorește să mulțumească și lui Thomas Karlsson, Colin McKinnon, Sasha Pachev, Eric Savage, Derick H. Siddoway și Bob Worthy pentru contribuția adusă la această carte.

Prefață

Când mi s-a oferit posibilitatea de a efectua o expertiză tehnică a cărții lui Paul, nu m-am așteptat să mă „înțepenesc" la prefață. Am fost întotdeauna de părere că un programator nu trebuie niciodată lăsat să scrie o carte, nici măcar una pentru programatori.

S-ar putea să am un oarecare talent pentru construirea unor linii de program utilizabile dar, din fericire, Paul are un mare talent de a explica lucruri pe care un mare număr de utilizatori le-ar înțelege altfel foarte greu. Aceia dintre noi care am lucrat la MySQL ne-am străduit din greu pentru a face ca acest limbaj să fie rapid și ușor de folosit, dar MySQL are numeroase caracteristici distincte, iar marea majoritate a utilizatorilor vor avea nevoie de informații care pot fi acumulate numai din experiență pentru a putea utiliza MySQL la maximum de potențial. În anul care a trecut, Paul a lucrat la manualul MySQL, proces în cursul căruia ne-a pus multe sute de întrebări privind modul de funcționare a limbajului MySQL, într-un efort de a înțelege totul. În această carte, el vă împărtășește și dumneavoastră cunoștințele acumulate. (De asemenea, sunteți bineveniți să lucrați cu manualul electronic de MySQL, chiar dacă acum aveți la dispoziție o modalitate mai rapidă de a înțelege MySQL.)

Manualul de MySQL a fost întotdeauna privit de noi, cei de la TcX, ca o referință tehnică pentru MySQL. Am avut întotdeauna nevoie de un manual de utilizare care să poată transmite conceptele tehnice într-un mod mai „relaxat". Credem că volumul de față va umple în mod elegant acest gol! Unul dintre scopurile limbajului MySQL este de a fi cât mai compatibil cu putință cu versiunile anterioare. Suntem siguri că, în ciuda ritmului rapid de dezvoltare al limbajului MySQL, majoritatea conceptelor din această carte își vor păstra veridicitatea pentru o lungă perioadă de timp.

Dacă mă mai bucur de atenția dumneavoastră, aș dori să mă folosesc de această ocazie pentru a mulțumi soției mele, Carola, pentru timpul pe care m-a lăsat să mi-l petrec cu MySQL în ultimii trei ani. Aș vrea să le mulțumesc și copiilor mei, Max și My, care au înțeles că, chiar dacă tatăl lor lucrează acasă, nu este întotdeauna disponibil. De asemenea, doresc să-i mulțumesc lui David pentru că m-a convins să scriu MySQL, precum și lui Allan Larson, care mi-a oferit posibilitatea de a face acest lucru. De asemenea, am avut parte de un mare ajutor din partea echipei actuale de MySQL: Jani Tolonen, Sinisa Milivojevic, Antti Halonen și Tonu Samuel. În final, dar nu în ultimul rând, doresc să le mulțumesc tuturor acelora care au scris interfețe API și programe client, care au trimis remedii și rapoarte privind unele defecte, care ne-au ajutat să transformăm MySQL în programul de succes care este astăzi.

Michael "Monty" Widenius
Helsingfors, 14 noiembrie 1999

Din partea autorului

Volumul de față a beneficiat în mare măsură de comentariile, corecturile și criticile furnizate de referenții tehnici: David Axmark, Vijay Chaugule, Chad Cunningham, Bill Gerrard, Jijo George John, Fred Read, Egon Schmid și Jani Tolonen. Mulțumiri speciale pentru Michael "Monty" Widenius, principalul creator al limbajului MySQL, care nu numai că a revizuit manuscrisul, dar a ridicat sute de întrebări la care am răspuns pe parcursul redactării cărții. Evident, toate erorile rămase îmi aparțin.

De asemenea, aș dori să mulțumesc lui Tomas Karlsson, Colin McKinnon, Sascha Pachev, Eric Savage, Derick H. Siddoway și Bob Worthy, care au examinat manuscrisul inițial și au ajutat la transpunerea acestei cărți în forma sa actuală.

Personalul editurii New Riders este responsabil în primul rând pentru conceperea acestei cărți și în al doilea rând pentru transformarea mâzgălelilor mele în opera finită pe care o aveți în față. Laurie Petrycki a fost editor executiv. Katie Purdum, editor responsabil cu achizițiile, m-a ajutat să-mi „intru în formă" și a preluat toate criticile atunci când nu reușeam să respect termenele. Leah Williams a fost nu numai editor de dezvoltare, dar și redactor de carte propriu-zis; a lucrat târziu de multe ori, mai ales în fazele finale ale proiectului. Cheryl Lenser și Tim Wright au produs indexul. John Rahm a fost editor de proiect. Debra Neel a corectat manuscrisul. Gina Rexrode și Wil Cruz, tehnoredactori, au aranjat cartea în forma pe care o vedeți acum. Mulțumiri fiecăruia dintre ei.

Cel mai mult, doresc să-mi exprim recunoștința față de soția mea, Karen, care s-a împăcat cu încă o carte, precum și pentru înțelegerea și răbdarea ei atunci când dispăream, uneori cu zilele, în „zona de scriere". Sprijinul ei mi-a ușurat munca în multe situații și sunt bucuros să-mi exprim recunoștința; ea m-a ajutat să scriu fiecare pagină a cărții.

Introducere

Un sistem de gestiune a bazelor de date relaționale (SGBDR) este un instrument esențial în numeroase medii, de la utilizările mai tradiționale în contexte de afaceri, cercetare și învățământ și până la aplicațiile mai recente, cum ar fi operarea motoarelor de căutare din Internet. Totuși, în ciuda importanței unei baze de date performante pentru gestiunea și accesul la resursele informaționale, aceasta s-a dovedit a fi dincolo de resursele financiare a numeroase instituții. Din punct de vedere istoric, sistemele de baze de date au constituit o propunere costisitoare, firmele distribuitoare percepând onorarii substanțiale, atât pentru program cât și pentru asistența necesară, iar deoarece motoarele de baze de date prezentau frecvent cerințe hardware substanțiale pentru a putea rula cu performanțe cât de cât rezonabile, costurile erau și mai mari.

În anii din urmă, situația s-a schimbat, atât din punct de vedere al echipamentelor, cât și din acela al programelor. Calculatoarele personale au devenit necostisitoare, dar puternice; pe de altă parte, a apărut o întreagă mișcare în direcția scrierii unor sisteme de operare cu performanțe ridicate pentru aceste calculatoare, sisteme disponibile la prețul unui compact disc ieftin, sau chiar gratuit, prin Internet. Acestea includ numeroase sisteme derivate din BSD UNIX (FreeBSD, NetBSD, OpenBSD), precum și diferite forme de Linux (RedHat, Caldera, LinuxPPC, pentru a numi doar câteva).

Producția de sisteme de operare gratuite care să permită utilizarea calculatoarelor personale la maximum de capacitate s-a desfășurat în mod concertat cu dezvoltarea unor instrumente disponibile gratuit, cum ar fi `gcc`, compilatorul GNU de C, fiind în mare măsură posibilă datorită acestora din urmă. Aceste eforturi de a pune programele la dispoziția oricărui doritor au avut ca rezultat ceea ce se numește acum mișcarea Open Source și au generat multe programe importante. Cel mai solicitat sit FTP din lume, și anume `ftp.cdrom.com`, rulează FreeBSD. Apache este cel mai folosit server Web din Internet. Alte succese ale inițiativei Open Source sunt limbajul de scripting de uz general Perl și PHP, un limbaj a cărui popularitate este într-o creștere rapidă, datorită ușurinței cu care permite scrierea paginilor Web dinamice. Toate acestea contrastează cu soluțiile „de firmă", care vă obligă să folosiți produse costisitoare, create de fabricanți care nici măcar nu furnizează codul sursă.

Programele de baze de date au devenit și ele mai accesibile. Sistemele de baze de date precum Postgres și mSQL au devenit disponibile gratuit sau la un preț scăzut. Mai recent, producătorii comerciali, precum Informix și Oracle, au început să-și ofere programele gratuit pentru sisteme de operare precum Linux. (Totuși, aceste din urmă produse sunt livrate, în general, numai în formă binară, fără suport, ceea ce le limitează utilitatea.)

Unul din noii veniți în domeniul bazelor de date cu preț scăzut sau gratuite este MySQL, un sistem client/server de gestiune a bazelor de date relaționale originar din Scandinavia. MySQL include un server SQL, programe client pentru accesul la server, instrumente administrative și o interfață de programare pentru scrierea propriilor dumneavoastră programe.

Bazele sistemului MySQL au fost puse în 1979, o dată cu instrumentul pentru baze de date UNIREG, creat de Michael "Monty" Widenius pentru compania suedeză TcX. În 1994, TcX a început să caute un server SQL pentru a-l utiliza la dezvoltarea aplicațiilor

Web. Compania a testat unele servere comerciale, dar toate s-au dovedit a fi prea lente pentru tabelele de mari dimensiuni ale firmei. De asemenea, compania a examinat mSQL, dar acestuia îi lipseau anumite caracteristici obligatorii pentru TcX. În consecință, Monty a început să programeze un server nou. Interfața de programare era proiectată în mod explicit pentru a fi similară celei folosite de mSQL, deoarece pentru mSQL erau disponibile numeroase instrumente gratuite, iar prin utilizarea unei interfețe similare aceleași instrumente puteau fi folosite pentru MySQL, cu un efort de portare minim.

În 1995, David Axmark de la Detron HB a început să facă presiuni pentru ca TcX să lanseze MySQL pe Internet. De asemenea, David lucra la documentație și la a determina MySQL să construiască folosind utilitarul GNU `configure`. MySQL 3.11.1 a fost dat lumii întregi în 1996, sub formă de distribuție binară pentru Linux și Solaris. În prezent, MySQL funcționează pe mult mai multe platforme și este disponibil atât în formă binară, cât și sursă.

MySQL nu este un proiect Open Source, deoarece este necesară o licență în anumite condiții. Totuși, MySQL se bucură de o amplă popularitate în comunitatea Open Source, deoarece termenii de licență nu sunt foarte restrictivi. (În esență, MySQL este în general gratuit, dacă nu doriți să obțineți profit prin vânzarea sistemului sau a unor servicii care necesită utilizarea acestuia.)

Popularitatea sistemului MySQL nu este limitată la comunitatea Open Source. Da, rulează pe calculatoare personale (într-adevăr, o bună parte din programarea cu MySQL are loc pe sisteme Linux ieftine). Dar MySQL este portabil și rulează pe sisteme de operare comerciale (precum Solaris, Irix și Windows) și pe echipamente care merg până la servere de întreprindere. În plus, performanțele sale rivalizează cu acelea ale oricărui sistem de baze de date cu care doriți să îl comparați și poate manipula baze de date de mari dimensiuni, cu milioane de înregistrări.

MySQL apare foarte clar în imaginea care se desfășoară dinaintea ochilor noștri: sisteme de operare disponibile gratuit, care rulează pe echipamente puternice, dar necostisitoare, punând la dispoziția unui număr de oameni mai mare ca oricând o putere substanțială de prelucrare a datelor și alte caracteristici, pe o varietate de sisteme mai largă ca oricând. Această „coborâre" a barierelor economice în ceea ce privește prelucrarea automată a datelor pune soluții puternice pentru baze de date la dispoziția unui număr fără precedent de mare de persoane și instituții. De exemplu, eu folosesc MySQL cu Perl, Apache și PHP pe calculatorul meu G3 PowerBook, care rulează LinuxPPC. Aceasta îmi permite să-mi transport oriunde rezultatele activității mele. Preț total: costul calculatorului PowerBook.

Instituții care în trecut se mărgineau să viseze la a exploata în folos propriu puterea unui SGBDR cu performanțe ridicate au acum această posibilitate, la un preț foarte redus. Utilizarea bazelor de date este tot mai frecventă și la nivel individual. Oameni care nu se gândeau niciodată că vor folosi baze de date încep să ia în considerare tot felul de utilizări ale acestora, din moment ce procurarea unui sistem de baze de date este facilă – de exemplu, stocarea și accesul la rezultatele unor cercetări genealogice, urmărirea și întreținerea colecțiilor de diferite tipuri (fluturi, mărci poștale, cărți de joc cu jucători de baseball etc.), asistență în demararea unei afaceri sau furnizarea de posibilități de căutare pentru situri Web personale.

De ce să optez pentru MySQL?

Dacă sunteți în căutarea unui sistem gratuit sau necostisitor de gestiune a bazelor de date, puteți alege din mai multe opțiuni: MySQL, mSQL, Postgres, unul din motoarele gratuite (dar fără suport) furnizate de producătorii comerciali și așa mai departe. Când comparați MySQL cu alte sisteme de baze de date, gândiți-vă la ceea ce este cel mai important pentru dumneavoastră. Performanță, suport, caracteristici (conformanță cu SQL, extensii etc.), condiții și restricții de licențiere, preț; toate acestea constituie factori care trebuie luați în considerare. Date fiind aceste considerații, MySQL are de oferit numeroase caracteristici atractive:

- **Viteză.** MySQL este rapid. Programatorii pretind că MySQL este cel mai rapid sistem de baze de date pe care îl puteți găsi. Puteți verifica această afirmație vizitând `http://www.mysql.com/benchmark.html`, o pagină de comparație a performanțelor din situl Web MySQL.
- **Ușurință în utilizare.** MySQL este un sistem de baze de date cu performanțe ridicate, dar relativ simplu, a cărui configurare și administrare sunt mult mai simple decât în cazul sistemelor mai mari.
- **Cost.** MySQL este gratuit pentru majoritatea utilizărilor interne. Vezi secțiunea „MySQL este gratuit?" ulterior în această introducere.
- **Suport pentru limbaje de interogare.** MySQL înțelege SQL (Structured Query Language – limbaj de interogare structurat), limbajul preferat al tuturor sistemelor moderne de baze de date. De asemenea, puteți avea acces la MySQL folosind aplicații care acceptă ODBC (Open Database Connectivity), un protocol de comunicație cu bazele de date creat de Microsoft.
- **Caracteristici.** La server se pot conecta mai mulți clienți simultan. Clienții pot folosi mai multe baze de date simultan. Puteți obține acces la MySQL în mod interactiv, folosind numeroase interfețe care vă permit să introduceți interogări și să vizualizați rezultate: clienți în linie de comandă, browsere Web sau clienți X Window System. De asemenea, este disponibilă o varietate de interfețe de programare pentru limbaje precum C, Perl, Java, PHP și Python. Astfel, aveți opțiunea de a folosi programe client preambalate sau de a vă scrie propriile programe client pentru aplicații personalizate.
- **Conectivitate și securitate.** MySQL poate fi folosit integral în rețele, iar bazele de date sunt accesibile de oriunde din Internet, deci vă puteți partaja datele cu oricine, oriunde. Dar MySQL are controlul accesului, astfel încât persoanele care nu au dreptul să vă citească datele nu vor avea această posibilitate.
- **Portabilitate.** MySQL rulează pe numeroase varietăți de UNIX, precum și pe alte sisteme non-UNIX, ca Windows și OS/2. MySQL rulează pe echipamente de la calculatoare de birou la servere cu performanțe ridicate.
- **Distribuție liberă.** MySQL este ușor de obținut; pentru aceasta, folosiți-vă browserul de Web. Dacă nu înțelegeți funcționarea unei componente sau sunteți curios cu privire la un algoritm, vă puteți procura codul sursă și îl puteți examina. Dacă nu vă place modul de funcționare a unei componente, îl puteți modifica.

Dar suportul? Bună întrebare; o bază de date nu este de prea mare folos dacă nu puteți obține asistență în raport cu ea. Natural, mi-ar plăcea să cred că această carte conține tot

ajutorul de care aveți nevoie. Realist vorbind, însă, veți avea întrebări la care nu m-am gândit niciodată sau pentru care nu am avut spațiu suficient. Veți descoperi că mai există și alte resurse, dar și că MySQL este dotat cu un sistem de asistență performant:

- MySQL include un manual de referință de mari dimensiuni (450 de pagini și mai crește...)
- Sunt disponibile contracte de asistență tehnică de la dezvoltatorii MySQL, pentru cei care preferă un acord formal.
- Există o listă de corespondență activă, la care se poate înscrie oricine. Această listă conține numeroși participanți utili, inclusiv dezvoltatorii MySQL. Ca resursă pentru asistență, mulți o găsesc suficientă pentru necesitățile proprii.

Comunitatea MySQL, programatori și ne-programatori, este extrem de cooperantă. Răspunsurile la întrebările din lista de corespondență sosesc, de regulă, în câteva minute. La semnalarea unor hibe, programatorii lansează un remediu în câteva zile (uneori ore!), iar remediile sunt disponibile imediat prin Internet. Comparați aceasta cu experiența deseori frustrantă a navigării prin canalele „bizantine“ de suport ale marilor producători. (Ați fost pe-acolo? Și eu am fost. Știu ce alternativă să folosesc atunci când am o întrebare despre un produs. Nu se poate face comparație între a fi pus pe „apel în așteptare“ la discreția unui producător și posibilitatea de a publica o întrebare într-o listă de corespondență și a verifica existența răspunsurilor atunci când îmi convine.)

MySQL este un candidat ideal pentru evaluare dacă vă aflați în procesul de selectare a sistemelor de baze de date. Puteți încerca MySQL fără nici un pericol de implicare financiară. Totuși, dacă vă încurcați, puteți folosi lista de corespondență pentru a fi ajutat. O evaluare costă o parte din timpul dumneavoastră, dar acest lucru este valabil indiferent de sistemul de baze de date pe care îl aveți în vedere – și este perfect adevărat că timpul de instalare și configurare pentru MySQL este mai redus decât pentru multe alte sisteme.

Folosiți deja un alt SGBDR?

Dacă în prezent utilizați un alt sistem de baze de date, trebuie să treceți la MySQL? Nu în mod necesar. Dacă sunteți satisfăcut de sistemul dumneavoastră actual, de ce să vă deranjați cu o atare tranziție? Dar dacă vă simțiți limitat de sistemul folosit, trebuie categoric să aveți în vedere MySQL. Poate că performanțele sistemului dumneavoastră curent reprezintă o problemă, sau poate că sistemul respectiv este „de firmă“ și nu vă place să fiți „blocat“ de acesta. Poate ați prefera să folosiți componente hardware care nu sunt acceptate de sistemul dumneavoastră actual sau poate că programul este furnizat în format binar, iar dumneavoastră preferați să aveți la dispoziție codul sursă. Sau poate că pur și simplu e prea costisitor! Toate acestea reprezintă motive pentru a lua în considerare MySQL. Folosiți această carte pentru a vă familiariza cu posibilitățile sistemului MySQL, puneți câteva întrebări în lista de corespondență MySQL și probabil că veți găsi răspunsurile necesare pentru a lua o decizie.

Dacă aveți în vedere trecerea de la un alt sistem de baze de date SQL la MySQL, examinați pagina de comparație din situl Web MySQL, la adresa `http://www.mysql.com/crash-me-choose.htmy`. Apoi, consultați capitolele care tratează tipurile de date MySQL și dialectul SQL. Puteți ajunge la concluzia că versiunea de SQL acceptată de sistemul dumneavoastră curent de gestiune a bazelor de date relaționale este mult prea diferit și că portarea aplicațiilor dumneavoastră ar implica un efort semnificativ.

O parte din evaluarea dumneavoastră trebuie să o reprezinte încercarea de a porta diferite exemple, desigur, deoarece s-ar putea să nu fie atât de greu precum credeți. S-ar putea să nu fie chiar atât de greu chiar dacă sistemul dumneavoastră de baze de date este unul mai vechi, care nu înțelege SQL. Eu am convertit recent un sistem de gestiune a înregistrărilor de la un SGBDR care nu era bazat pe SQL. Nu exista nici o asemănare între limbaje care să poată fi utilizată, iar unele dintre tipurile de date nu aveau echivalent în SQL. Acest proces a necesitat conversia metodelor de acces la rețea și zeci de programe de introducere bazate pe ecran și de interogări „conservate“. A fost necesară o lună și jumătate de eforturi „cu normă întreagă“, ceea ce n-a fost chiar atât de rău.

Instrumente furnizate cu MySQL

Distribuția MySQL include următoarele instrumente:

- **Un server SQL.** Acesta este motorul care activează MySQL și care furnizează accesul la bazele dumneavoastră de date.
- **Programe client pentru accesul la server.** Un program interactiv vă permite să introduceți interogări în mod direct și să vizualizați rezultatele, iar numeroasele programe administrative și utilitare vă ajută să vă rulați situl. Un utilitar vă permite să controlați serverul. Altele vă permit să importați sau să exportați date, să verificați permisiunile de acces și multe altele.
- **O bibliotecă client pentru scrierea propriilor dumneavoastră programe.** Puteți scrie programe client în C, deoarece biblioteca este scrisă în C, dar biblioteca mai furnizează și baza pentru terțe asocieri pentru alte limbaje.

În afara programelor furnizate cu MySQL, MySQL însuși este folosit de către numeroase persoane talentate și capabile, cărora le place să scrie programe pentru a-și îmbunătăți productivitatea și care doresc să pună la dispoziția publicului aceste programe. Rezultatul este că aveți acces la o diversitate de instrumente produse de terțe părți, care facilitează utilizarea sistemului MySQL sau care extind aria de acțiune a acestuia în domenii precum dezvoltarea siturilor Web.

MySQL este gratuit?

MySQL nu este un produs Open Source, dar, în general, poate fi folosit gratuit. Trebuie să consultați manualul de referință MySQL pentru detalii privind licența, dar principiile de bază sunt următoarele:

- Programele client și biblioteca de programare a clienților sunt gratuite, pe toate platformele.
- Pentru UNIX și alte platforme non-Windows, serverul MySQL poate fi folosit gratuit, cu excepția situațiilor când doriți să vindeți serverul sau alte programe sau servicii care impun utilizarea acestuia. În această situație, trebuie să obțineți licență pentru server. Ideea este că, dacă obțineți un profit din MySQL, este normal ca dezvoltatorii sistemului să primească o parte din acesta. (200 de dolari este o nimica toată pentru un specialist SGBDR care vă ajută să obțineți un profit și există o mulțime de programe gratuite pe care vi le puteți procura pentru a folosi eficient sistemul.)

- Versiunea Windows a serverului MySQL impune obţinerea unei licenţe.
- Versiunile mai vechi de MySQL sunt disponibile în condiţiile licenţei publice GNU (GPL) şi pot fi folosite în orice scopuri, fără nici o plată. MySQL 3.20.32a este disponibil în condiţiile GPL.

Indiferent dacă aveţi sau nu nevoie de o licenţă pentru server, se poate obţine o asistenţă formală, contra cost, de la programatorii MySQL, şi vă sfătuiesc să folosiţi această ocazie, mai ales dacă lucraţi într-o firmă a cărei conducere „mârâie" la fiecare program pentru care nu există un astfel de acord. Sunt disponibile numeroase niveluri de suport şi, în afară de obţinerea unei asistenţe excelente, veţi contribui la dezvoltarea în continuare a sistemului MySQL şi, implicit, a întregii comunităţi MySQL.

O notă despre licenţa Windows

Uneori se pune întrebarea: „Cum adică trebuie să obţin licenţă pentru serverul MySQL numai fiindcă rulez Windows?" Aceasta este o întrebare rezonabilă, la care există un răspuns rezonabil. Gândiţi-vă la aspectele economice ale problemei. Pentru a vă angaja în dezvoltarea de programe, aveţi nevoie de un sistem de operare şi de unele instrumente de dezvoltare, cum ar fi un editor şi un compilator. Relativ la aceste cerinţe, există o diferenţă fundamentală între dezvoltarea pentru UNIX şi dezvoltarea pentru Windows. Sub UNIX, puteţi obţine gratuit cam tot ceea ce aveţi nevoie:

- Pentru sistemul de operare, aveţi numeroase opţiuni, cum sunt Linux sau una din variantele BSD gratuite, cum sunt FreeBSD, NetBSD şi OpenBSD.
- Pentru instrumentele de dezvoltare, toate aceste sisteme de operare sunt dotate cu editoare precum vi şi emacs, precum şi compilatoare ca gcc şi egcs.
- Când sunt lansate actualizări ale sistemelor de operare sau ale instrumentelor de dezvoltare, le puteţi pur şi simplu descărca din Internet sau vă puteţi procura un compact disc necostisitor. Acest lucru este valabil chiar şi pentru revizuirile substanţiale.

Pe de altă parte, dezvoltarea programelor în Windows este relativ costisitoare:

- Sistemul de operare nu este gratuit.
- Instrumentele de dezvoltare, precum compilatorul, nu sunt gratuite.
- Când sunt lansate actualizări ale sistemului de operare sau ale instrumentelor, trebuie să plătiţi din nou, chiar dacă acestea nu sunt altceva decât remedii pentru hibe sau actualizări incrementale de mică importanţă.

Toate acestea înseamnă că, în timp ce costurile pentru dezvoltarea programelor în UNIX sunt, în esenţă, nule, pentru Windows acestea pot atinge valori considerabile. Se prea poate ca dezvoltatorilor MySQL să le placă să lucreze cu MySQL, dar nu atât de mult încât să fie dispuşi să plătească pentru acest privilegiu. Costurile de dezvoltare sub Windows trebuie recuperate într-un fel sau altul, iar taxa pentru licenţă este modul în care se produce această recuperare.

În plus, dezvoltatorii au descoperit că programarea sub Windows necesită mai mult timp decât cea sub UNIX. Deoarece timpul dezvoltatorilor are o mărime fixă, acesta este un cost care trebuie suportat pe spezele versiunii UNIX (care este, în fond, platforma primară de dezvoltare a sistemului MySQL). Taxa de licenţă Windows furnizează un stimulent pentru orientarea timpului şi a eforturilor spre portul Windows.

Dacă doriţi să rulaţi Windows, dar nu doriţi să obţineţi licenţă pentru server, aveţi unele opţiuni:

- Există o versiune shareware a sistemului MySQL, pe care o puteţi testa pentru evaluare. Această versiune vă oferă 30 de zile pentru utilizarea serverului, după care puteţi decide dacă veţi plăti licenţa pentru el.
- Dacă rulaţi serverul Windows în scopuri educaţionale sau în instituţii universitare sau de cercetare guvernamentale, puteţi cere dezvoltatorilor să vă scutească de taxa de licenţă.
- Programele client sunt gratuite în orice caz, deci, dacă puteţi găsi pe cineva care rulează deja un server şi este dispus să vă permită să-l folosiţi, aveţi acces complet la caracteristicile sistemului MySQL.

Ce puteţi aştepta de la această carte

Citind volumul de faţă, puteţi învăţa să folosiţi sistemul MySQL în mod eficient, astfel încât să vă puteţi face treaba într-un mod mai productiv. Veţi putea determina modul de introducere a informaţiilor dumneavoastră într-o bază de date şi veţi învăţa să formulaţi interogări care să vă ofere răspunsuri la întrebări pe care doriţi să le puneţi relativ la acele date.

Nu trebuie să fiţi programator pentru a înţelege sau pentru a folosi SQL. Această carte vă va arăta cum funcţionează sistemul. Dar, pentru a înţelege modul de utilizare adecvată a unei baze de date, nu este suficient să cunoşti sintaxa SQL. Lucrarea pune accentul pe caracteristicile unice ale sistemului MySQL şi indică modul de utilizare a acestora. Alte cărţi se concentrează asupra limbajului SQL în general, respectiv asupra dialectului SQL al unui alt SGBDR.

De asemenea, veţi vedea care este modul de integrare al sistemului MySQL cu alte instrumente. Cartea prezintă modul de utilizare a sistemului MySQL cu PHP sau Perl pentru a genera pagini Web dinamice, create din rezultatele interogărilor aplicate bazelor de date. Veţi învăţa să scrieţi propriile dumneavoastră programe care să aibă acces la bazele de date MySQL. Toate acestea îmbunătăţesc posibilităţile SQL de a satisface cerinţele aplicaţiilor dumneavoastră particulare.

Dacă veţi fi responsabil cu administrarea unei instalări MySQL, această carte vă va arăta care sunt îndatoririle dumneavoastră şi modul de îndeplinire a acestora. Veţi învăţa să configuraţi conturi de utilizator, să executaţi copii de siguranţă ale bazelor de date şi să luaţi măsuri pentru a asigura securitatea sitului dumneavoastră.

„Harta rutieră" a acestei cărţi

Lucrarea de faţă este structurată în patru părţi.

Partea I: Utilizarea generală a sistemului MySQL

- **Introducere în MySQL şi SQL.** Discută despre modul în care MySQL vă poate fi de folos şi conţine un manual general care prezintă programul client interactiv MySQL şi care tratează elementele fundamentale ale limbajului SQL.
- **Lucrul cu date în MySQL.** Discută tipurile de coloane furnizate de SQL pentru descrierea datelor dumneavoastră, proprietăţile şi dezavantajele fiecărui tip, când şi cum să utilizaţi aceste tipuri, modul de selectare dintre tipuri similare, evaluarea expresiilor şi conversia tipurilor.
- **Sintaxa şi utilizarea SQL în MySQL.** Fiecare SGBDR important existent în prezent înţelege SQL, dar fiecare motor de baze de date implementează un dialect SQL uşor diferit. Acest capitol discută despre SQL, cu accent asupra acelor caracteristici distinctive ale sistemului MySQL. De asemenea, discută despre caracteristici prezente în alte sisteme de baze de date, dar inexistente în SQL, precum şi soluţiile existente.
- **Optimizarea interogărilor.** Un mod mai eficient de rulare a interogărilor.

Partea a II-a: Utilizarea interfeţelor de programare ale sistemului MySQL

- **Introducere în programarea MySQL.** Discută despre unele interfeţe de programare a aplicaţiilor disponibile sub MySQL şi conţine o comparaţie generală a interfeţelor API pe care cartea le tratează în detaliu.
- **Interfaţa API MySQL pentru C.** Modul de scriere a programelor C folosind interfaţa API furnizată de biblioteca client inclusă în distribuţia MySQL.
- **Interfaţa API pentru Perl DBI.** Modul de scriere a scripturilor Perl folosind modulul DBI. Tratează scripturile autonome şi scripturile CGI pentru programarea siturilor Web.
- **Interfaţa API pentru PHP.** Modul de utilizare a limbajului de scripting PHP pentru a scrie pagini Web dinamice, care obţin acces la bazele de date MySQL.

Partea a III-a: Administrarea sistemului MySQL

- **Introducere în administrarea sistemului MySQL.** Prezintă îndatoririle unui administrator de baze de date şi noţiunile pe care trebuie să le cunoaşteţi pentru a rula un sit performant.
- **Catalogul de date MySQL.** O examinare aprofundată a organizării şi conţinutului catalogului de date, regiunea în care MySQL stochează bazele de date şi fişierele de stare.
- **Administrarea generală a sistemului MySQL.** Modul în care vă puteţi asigura că serverul dumneavoastră porneşte, respectiv se opreşte în mod adecvat atunci când sistemul dumneavoastră execută această operaţie. De asemenea, include instrucţiuni pentru configurarea conturilor de utilizator MySQL şi discută despre întreţinerea fişierelor jurnal, despre strategiile de creare a copiilor de siguranţă, despre ajustarea serverelor şi despre strategiile de realizare a copiilor de siguranţă, respectiv de refacere a bazelor de date.
- **Securitate.** Ceea ce trebuie să ştiţi pentru a proteja instalarea sistemului dumneavoastră MySQL împotriva intruşilor, atât contra altor utilizatori ai gazdei serverului, cât şi a clienţilor care se conectează prin reţea.
- **Întreţinerea şi repararea bazelor de date.** Modul de a reduce posibilitatea de apariţie a dezastrelor printr-o întreţinere preventivă, precum şi metodele de a efectua refacerea în urma căderilor, dacă dezastrele survin în ciuda măsurilor dumneavoastră de prevenire.

Partea a IV-a: Anexe

- **Obţinerea şi instalarea programelor.** De unde să vă procuraţi principalele instrumente descrise în carte şi modul de instalare a acestora.
- **Referinţă de tipuri de coloane.** Descrierea tipurilor de coloane MySQL.
- **Referinţă de operatori şi funcţii.** Descrierea operatorilor şi a funcţiilor care se pot folosi pentru a scrie expresii în instrucţiunile SQL.
- **Referinţă de sintaxă SQL.** Descrierea fiecărei instrucţiuni SQL pe care MySQL o poate înţelege.

- **Referinţă de programe MySQL.** Descrierea fiecărui program furnizat în distribuţia MySQL.
- **Referinţă API C.** Descrierea tipurilor de date şi a funcţiilor din biblioteca client C a sistemului MySQL.
- **Referinţă API Perl DBI.** Descrierea metodelor şi a atributelor furnizate de modulul Perl DBI.
- **Referinţă API PHP.** Descrieri ale funcţiilor furnizate de PHP pentru suportul MySQL.
- **Instrumente utile produse de terţe părţi.** O succintă trecere în revistă a unora dintre instrumentele disponibile pentru a vă ajuta să utilizaţi MySQL, cum sunt utilitarele de conversie a datelor sau utilitarele administrative.
- **Furnizori de servicii Internet.** Ce anume trebuie avut în vedere atunci când alegeţi un ISP care furnizează acces MySQL. Ce anume trebuie avut în vedere atunci când funcţionaţi ca furnizor de servicii Internet care asigură clienţilor servicii MySQL.

Cum să citiţi această carte

Indiferent de porţiunea din carte pe care o citiţi la un moment dat, cel mai bine este să încercaţi exemplele pe măsură ce parcurgeţi cartea. Dacă MySQL nu este instalat în sistemul dumneavoastră, trebuie să îl instalaţi sau să rugaţi pe cineva să îl instaleze. Apoi, procuraţi-vă fişierele necesare pentru configurarea bazei de date demonstrative la care vom face referire pe parcursul cărţii. Anexa A, „Obţinerea şi instalarea programelor", vă arată de unde vă puteţi procura toate acestea şi conţine instrucţiuni pentru instalarea lor.

Dacă nu aveţi nici cele mai elementare noţiuni privind MySQL sau SQL, începeţi de la capitolul 1, „Introducere în MySQL şi SQL". Acest capitol vă oferă o introducere de tip manual care vă prezintă fundamentele conceptelor privind MySQL şi SQL şi vă oferă cele necesare pentru a putea parcurge restul cărţii. Apoi, treceţi la capitolul 2, „Lucrul cu date în MySQL", respectiv capitolul 3, „Sintaxa şi utilizarea SQL în MySQL", pentru a descoperi modul de descriere şi manipulare a datelor dumneavoastră, astfel încât să puteţi exploata caracteristicile sistemului MySQL pentru propriile dumneavoastră aplicaţii.

Dacă aveţi unele noţiuni de SQL, trebuie totuşi să citiţi capitolele 2 şi 3. Implementările SQL diferă şi veţi dori să ştiţi care sunt elementele distinctive ale implementării MySQL în raport cu alte implementări pe care, poate, le cunoaşteţi.

Dacă aveţi experienţă cu MySQL, dar aveţi nevoie de mai multe informaţii privind detaliile de efectuare a anumitor operaţii, puteţi folosi cartea ca pe o referinţă, căutând subiectele dorite în funcţie de necesităţi.

Dacă sunteţi interesat în a vă scrie propriile dumneavoastră programe pentru accesul la bazele de date MySQL, citiţi capitolele referitoare la API, începând de la capitolul 5, „Introducere în programarea MySQL". Dacă doriţi să generaţi o interfaţă bazată pe Web la bazele dumneavoastră de date pentru a facilita accesul la acestea sau, invers, pentru a furniza un back end de baze de date pentru situl dumneavoastră de Web în scopul de a vă îmbunătăţi situl cu un conţinut dinamic, examinaţi capitolele 7, „Interfaţa API pentru Perl DBI", respectiv 8, „Interfaţa API pentru PHP".

Dacă evaluați MySQL pentru a-l raporta la sistemul dumneavoastră curent de gestiune a bazelor de date relaționale, numeroase porțiuni ale cărții vă vor fi de ajutor. Citiți capitolele referitoare la tipurile de date și la sintaxa SQL din Partea I pentru a compara MySQL cu sistemul SQL cu care sunteți obișnuit, capitolele de programare din Partea a II-a dacă aveți aplicații personalizate, respectiv capitolele despre administrare din Partea a III-a pentru a estima nivelul de suport administrativ necesar unei instalări a sistemului MySQL. Aceste informații sunt de asemenea utile dacă nu folosiți în mod curent o bază de date, dar efectuați o analiză comparativă a sistemului MySQL și a altor sisteme de baze de date, în scopul de a alege unul din ele.

Dacă doriți acces la MySQL și căutați un furnizor de servicii Internet (ISP) care să-l ofere, vezi Anexa J, „Furnizori de servicii Internet", pentru sugestii privind alegerea unui furnizor. De asemenea, această anexă conține sfaturi pentru furnizorii de servicii care doresc să asigure MySQL pentru a-și atrage noi clienți sau pentru a-i servi mai bine pe clienții actuali.

Versiuni de programe tratate în această carte

În momentul scrierii acestor rânduri, versiunea curent stabilă de MySQL este seria de versiuni 3.22, iar seria 3.23 se află într-un proces de dezvoltare activă. Cartea se ocupă de ambele serii. Acolo unde anumite caracteristici nu există în versiunile anterioare (inclusiv seria 3.21), acest lucru este semnalat.

Pentru celelalte pachete principale de programe discutate aici, versiunile recente sunt suficiente pentru exemplele date în carte. Versiunile curente sunt:

Pachet	Versiune
Perl DBI	1.13
Driver Perl DBI pentru MySQL	1.22.xx (stabilă), 1.23.xx (beta)
PHP	3.0.12 (4.0 se află la orizont)
Apache	1.3.9
CGI.pm	2.56

Toate programele discutate în această carte sunt disponibile din Internet. Anexa A furnizează instrucțiuni pentru încărcarea în sistemul dumneavoastră a sistemului MySQL, a suportului pentru modulul Perl DBI, a limbajului PHP, a serverului Apache și a limbajului CGI.pm. Această anexă mai conține instrucțiuni pentru procurarea bazei de date demonstrative folosite în exemplele din carte, precum și a programelor exemplu create în capitolele de programare.

Convenții folosite în carte

Convențiile tipografice folosite sunt următoarele:

Corpul de literă monospațiat indică nume de gazde, nume de fișiere, nume de cataloage, comenzi, opțiuni și situri Web.

Acolo unde comenzile sunt indicate atunci când le introduceți, caracterele aldine indică textul pe care îl introduceți. Promptul indică modul de rulare a comenzii:

`%`	Comanda este rulată ca utilizator UNIX obișnuit
`#`	Comanda este rulată ca utilizator UNIX rădăcină (`root`)
`C:\>`	Comanda este rulată în Windows

Corpul de literă cu caractere italice din comenzi indică locul unde trebuie să introduceți o valoare la libera dumneavoastră alegere.

În instrucțiunile SQL, cuvintele cheie și numele funcțiilor SQL sunt scrise cu caractere majuscule. Numele bazelor de date, ale tabelelor și ale coloanelor sunt scrise cu caractere minuscule. În descrierile sintactice, parantezele drepte ([ ]) indică informații opționale.

Resurse suplimentare

Această carte încearcă să vă spună practic tot ceea ce trebuie să știți despre MySQL. Dar dacă aveți o întrebare al cărei răspuns nu se găsește în carte, încotro trebuie să vă îndreptați?

Între resursele utile sunt incluse siturile de Web pentru programele în raport cu care aveți nevoie de ajutor:

Pachet	Sit Web principal
MySQL	`http://www.mysql.com/doc.html`
Perl DBI	`http://www.symbolstone.org/technology/perl/DBI/`
PHP	`http://www.php.net/`
Apache	`http://www.apache.org/`
CGI.pm	`http://stein/cshl.org/WWW/software/CGI/`

Utilizarea manualului electronic de referință MySQL

Nu uitați să examinați ocazional manualul electronic de referință MySQL pentru informații privind cele mai recente îmbunătățiri referitoare la MySQL. Manualul va fi modificat continuu, pe măsura apariției de noi schimbări.

Aceste situri conțin indicatori spre diferite forme de informație, cum ar fi manualele de referință, liste cu întrebări frecvente (FAQ) și liste de corespondență:

- Manuale de referință

  Principala documentație inclusă chiar în SQL este manualul de referință. Acesta este disponibil în numeroase formate, inclusiv o versiune electronică.

  Manualul PHP este disponibil, de asemenea, în mai multe versiuni; totuși, versiunea electronică tinde să fie mai completă decât versiunea în format tipărit.

  Modulul DBI și driverul său specific pentru MySQL au o documentație separată. Documentul DBI conține concepte de ordin general. Documentul driverului MySQL discută caracteristici specifice sistemului MySQL.

- Întrebări frecvente (FAQ)

  Există liste FAQ pentru DBI, PHP și Apache.

- Liste de corespondență

  Există numeroase liste de corespondență orientate spre programele discutate în această carte. Este o idee bună să vă abonați la cele care se referă la instrumentele pe care doriți să le utilizați. Vă rugăm să nu uitați, totuși, că trebuie să învățați să folosiți arhivele (pentru listele care dispun de asemenea arhive). Dacă sunteți începător în utilizarea unui instrument, veți avea multe întrebări care au fost puse (și la care s-a răspuns) de un milion de ori, deci nu aveți nici un motiv să întrebați din nou dacă puteți găsi răspunsul printr-o căutare rapidă prin arhive.

  Instrucțiunile pentru a vă abona la listele de corespondență variază, dar puteți găsi informații la adresele URL prezentate aici:

Pachet	Instrucțiuni pentru lista de corespondență
MySQL	`http://www.mysql.com/doc.html`
Perl DBI	`http://www.symbolstone.org/technology/perl/DBI/`
PHP	`http://www.php.net/support.php3`
Apache	`http://www.apache.org/foundation/mailinglists.html`

- Situri Web subordonate

  În afara siturilor Web oficiale, unele dintre instrumentele discutate aici dispun de situri subordonate, care furnizează informații suplimentare, cum ar fi cod sursă demonstrativ sau articole centrate pe anumite subiecte. Căutați o regiune "Links" (legături) din situl oficial pe care îl vizitați.

PARTEA I

Utilizarea generală a sistemului MySQL

CAPITOLUL 1

Introducere în MySQL și SQL

Acest capitol constituie o introducere în sistemul de gestiune a bazelor de date relaționale (SGBDR) MySQL și în limbajul de interogare structurat (SQL) pe care îl înțelege sistemul MySQL. Capitolul prezintă termeni și concepte fundamentale pe care trebuie să le înțelegeți, descrie baza de date demonstrativă pe care o vom folosi în exemplele din carte și conține un manual care vă indică modul de utilizare a sistemului MySQL pentru a crea o bază de date și pentru a interacționa cu aceasta.

Începeți de aici dacă sunteți un neofit în domeniul bazelor de date și, poate, nu sunteți sigur dacă aveți nevoie sau dacă puteți folosi o bază de date. De asemenea, trebuie să citiți capitolul de față dacă nu știți nimic despre MySQL sau SQL și aveți nevoie de un ghid introductiv pentru început. Cititorii cu experiență în MySQL sau cu sistemele de baze de date pot omite acest material. Totuși, toată lumea trebuie să citească secțiunea „O bază de date demonstrativă", deoarece este cel mai bine să fiți familiarizat cu destinația și conținutul bazei de date pe care o vom folosi în mod repetat pe parcursul cărții.

Cum vă poate ajuta MySQL

Această secțiune descrie situații în care sistemul de baze de date MySQL poate fi util. Astfel, vă veți forma o părere cu privire la tipurile de operații pe care le poate efectua MySQL și la modalitățile în care acest sistem vă poate ajuta. Dacă nu trebuie să vă convingeți de utilitatea unui sistem de baze de date – poate fiindcă deja aveți o problemă în minte și doriți pur și simplu să aflați care este modul în care puteți folosi MySQL pentru a vă ajuta să o rezolvați – puteți trece la secțiunea „O bază de date demonstrativă".

Un *sistem de baze de date* este, în esență, o modalitate de a gestiona liste cu informații. Informațiile pot proveni dintr-o varietate de surse. De exemplu, pot reprezenta date de cercetare, înregistrări de afaceri, cereri ale clienților, statistici sportive, rapoarte de vânzări, informații despre hobby-urile proprii, înregistrări de personal, rapoarte privind defecțiuni sau notele unor elevi. Totuși, deși sistemele de baze de date pot lucra cu o gamă largă de informații, nu folosiți un asemenea sistem doar de dragul de a-l folosi. Dacă o sarcină este de la bun început ușor de îndeplinit, nu aveți nici un motiv de a insera o bază de date numai pentru a o folosi. O listă de cumpărături este un exemplu bun: scrieți articolele pe care trebuie să le cumpărați, le ștergeți de pe listă atunci când le cumpărați și apoi aruncați lista. Este puțin probabil să folosiți o bază de date pentru asta. Chiar dacă aveți un calculator de tip palmtop (care poate fi ținut în palmă – N.T.) probabil că pentru lista de cumpărături veți folosi funcția sa de editare de texte, nu caracteristicile sale de baze de date.

Puterea unui sistem de baze de date intră în acțiune atunci când informațiile pe care doriți să le organizați și să le gestionați devin voluminoase sau complexe, astfel încât înregistrările dumneavoastră devin prea dificil de manipulat manual. Bazele de date pot fi folosite, desigur, de către mari corporații, care prelucrează milioane de tranzacții zilnic. Dar chiar și operațiile la scară mică, care necesită o singură persoană pentru a întreține informații de interes personal, pot necesita o bază de date. Nu este greu să ne gândim la situații în care utilizarea unei baze de date poate fi benefică, deoarece nu trebuie să aveți cantități imense de informații înainte ca acele informații să devină dificil de gestionat. Să ne gândim la următoarele situații:

- Întreprinderea dumneavoastră de tâmplărie are numeroși angajați. Trebuie să păstrați înregistrările cu salariați și statele de plată astfel încât să știți pe cine ați plătit și când ați efectuat plata; de asemenea, trebuie să rezumați aceste înregistrări pentru a putea remite declarațiile de venit autorităților, în vederea stabilirii impozitelor. De asemenea, trebuie să păstrați evidența sarcinilor pe care compania dumneavoastră a fost angajată să le îndeplinească și salariații pe care i-ați programat să lucreze la fiecare sarcină.
- Sunteți administratorul unei rețele de depozite cu piese de schimb pentru autoturisme și trebuie să știți care sunt depozitele care dispun de o anumită piesă în inventar, astfel încât să puteți onora comenzile clienților.
- Ca vânzător de jucării, sunteți supus cu precădere unei cereri (dependente de capriciile cumpărătorilor) de articole pe care le puteți transporta. Doriți să știți care este traiectoria curentă a vânzărilor pentru anumite articole, astfel încât să puteți estima dacă să măriți inventarul (pentru un articol care devine mai popular) sau dacă să-l reduceți (pentru a nu vă „înțepeni" cu un stoc foarte mare dintr-un produs care nu se mai vinde bine).
- Stiva de date de cercetare pe care le-ați adunat în cursul mai multor ani trebuie să fie analizată în vederea publicării, pentru ca nu cumva dictonul „publică sau dispari" să devină epitaful carierei dumneavoastră. Doriți să reduceți mari cantități de date brute pentru a genera informații de sumar, precum și să extrageți subseturi selectate de observații în vederea unei analize statistice mai detaliate.
- Sunteți un orator cunoscut, care călătorește prin țară pentru a participa la numeroase tipuri de adunări, cum sunt festivitățile de absolvire, întâlnirile de afaceri, întruniri civice și convenții politice. V-ați adresat publicului de atâtea ori, încât este dificil să vă amintiți ce anume ați spus în fiecare loc unde ați fost, deci ați dori să păstrați înregistrări ale discursurilor dumneavoastră trecute și să le folosiți pentru a vă ajuta în planificarea angajamentelor dumneavoastră viitoare. Dacă reveniți într-un loc unde ați mai vorbit, nu doriți să țineți un discurs similar cu unul pe care l-ați mai ținut în acel loc, iar o înregistrare a fiecărui loc pe unde ați fost vă va ajuta să evitați repetările. De asemenea, doriți să rețineți modul în care au fost receptate discursurile dumneavoastră. (Monologul dumneavoastră „De ce-mi plac pisicile" de la Metropolitan Kennel Club a fost oarecum un fiasco, deci nu vreți să faceți din nou aceeași greșeală, dacă reveniți acolo.)
- Sunteți un profesor care trebuie să-și păstreze evidența notelor date și a prezenței la curs. De fiecare dată când dați un test sau un chestionar, înregistrați nota obținută de

fiecare elev. Este destul de simplu să scrieți notele într-un catalog, dar utilizarea ulterioară a notelor este o treabă dificilă. Preferați să evitați sortarea notelor de la fiecare test pentru a determina distribuția notelor; de asemenea, preferați să nu adunați punctajele fiecărui elev atunci când determinați mediile finale de la sfârșitul perioadei de notare. Nici numărarea absențelor fiecărui elev nu are nici un haz.

- Instituția al cărei secretar sunteți folosește un catalog al membrilor. (Instituția poate avea orice profil: o societate profesională, un club, o companie cu depozite, o orchestră simfonică sau un club pentru întreținerea condiției fizice.) În fiecare an, generați acest catalog în formă scrisă pentru membri, în funcție de un document creat cu un procesor de texte pe care îl editați atunci când informațiile despre membri se modifică.

  V-ați plictisit să folosiți catalogul în acest mod, deoarece limitează modurile de utilizare a acestuia. Sortarea intrărilor în diferite moduri este dificilă și nu puteți selecta cu ușurință numai anumite părți ale fiecărei intrări (cum ar fi o listă alcătuită numai din nume și numere de telefon). Nici nu puteți descoperi ușor un subset de membri, cum sunt aceia care trebuie să-și plătească în curând cotizația; dacă ați putea, ați elimina necesitatea de a căuta lunar în diferite rubrici pentru a-i găsi pe acei membri cărora trebuie să li se trimită înștiințări pentru plata cotizației.

  De asemenea, doriți efectiv să evitați să editați personal catalogul, dar bugetul societății este limitat, iar angajarea unei alte persoane este în afara oricărei discuții. Ați auzit despre „biroul fără hârtii" care ar rezulta din întreținerea electronică a înregistrărilor, dar încă nu ați „simțit" nici un avantaj al acestuia. Înregistrările pentru fiecare membru sunt în format electronic dar, ironic, nu se află într-o formă care poate fi folosită cu ușurință în orice scop, *mai puțin* generarea de hârtii prin tipărirea catalogului membrilor!

Aceste scenarii variază de la situații care implică mari cantități de informații și până la cazuri cu un volum de informații relativ redus. Elementul comun este existența unor operații care pot fi efectuate manual, dar care pot fi executate mai eficient de către un sistem de baze de date.

Care sunt avantajele concrete pe care trebuie să le așteptați din utilizarea unui sistem de baze de date precum MySQL? Aceasta depinde de necesitățile și cerințele dumneavoastră specifice, iar acestea sunt extrem de variate, așa cum s-a arătat în exemplele precedente. Să examinăm un tip de situație care survine frecvent și care astfel este destul de reprezentativ pentru utilizarea bazelor de date.

Sistemele de gestiune a bazelor de date sunt frecvent utilizate pentru executarea unor operații cum sunt cele pentru care se folosesc fișierele (nu cele electronice – N.T.). Într-adevăr, o bază de date este un fișier de mari dimensiuni din anumite puncte de vedere, dar un fișier cu un sistem de fișare încorporat. Păstrarea prin metode electronice a înregistrărilor prezintă câteva avantaje importante față de păstrarea manuală a înregistrărilor. De exemplu, dacă lucrați într-un birou în care sunt păstrate înregistrările clienților, iată câteva modalități în care MySQL vă poate ajuta:

- **Timp redus de fișare a înregistrării.** Nu trebuie să căutați în sertarele fișierelor pentru a determina unde să puneți o nouă înregistrare. „Înmânați" înregistrarea sistemului de fișare și îi permiteți să plaseze înregistrarea la locul potrivit pentru dumneavoastră.

- **Timp redus de regăsire a înregistrării.** Când căutați înregistrări, nu căutați personal în fiecare înregistrare pentru a le găsi pe cele care conțin informațiile dorite. Să presupunem că lucrați într-un cabinet stomatologic. Dacă doriți să trimiteți înștiințări tuturor pacienților care nu s-au prezentat la control de mai mult timp, cereți sistemului de fișare să găsească înregistrările adecvate. Desigur, veți proceda altfel decât dacă ați vorbi cu o altă persoană. Acesteia i-ați spune: „Fii amabil și află care sunt pacienții care nu s-au prezentat la control în ultimele 6 luni." În fața unei baze de date, emiteți o incantație bizară:

```
SELECT nume, prenume, ultima_vizita FROM pacient
WHERE ultima_vizita<DATE_SUB(CURRENT_DATE, INTERVAL 6 MONTH)
```

Această expresie vă poate intimida dacă nu ați mai văzut-o niciodată, dar perspectiva de a obține rezultate în câteva secunde, fără a mai căuta o oră încheiată, ar trebui să fie atrăgătoare. (În orice caz, nu vă îngrijorați. Bălmăjeala aia stranie nu vă va mai părea ciudată pentru multă vreme. De fapt, veți înțelege exact ce înseamnă când veți termina de citit acest capitol.)

- **Ordine de regăsire flexibilă.** Nu trebuie să regăsiți înregistrări în funcție de ordinea fixă în care le stocați (în funcție de numele de familie al pacientului, de exemplu). Puteți cere sistemului de fișare să extragă înregistrări sortate în orice ordine preferați: după numele de familie, numele companiei de asigurări, data ultimei vizite și altele.
- **Format flexibil al datelor de ieșire.** După ce ați descoperit înregistrările care vă interesează, nu trebuie să copiați informațiile manual. Puteți cere sistemului de fișare să genereze automat o listă. Uneori doriți numai să tipăriți informațiile. Alteori, doriți să le folosiți într-un alt program. (De exemplu, după ce generați lista cu pacienții întârziați la control, puteți insera această informație într-un procesor de texte care tipărește note pe care le puteți trimite acestor pacienți.) Sau vă interesează numai informații de sumar, cum ar fi numărul înregistrărilor selectate. Nu trebuie să le numărați personal; sistemul de fișare poate genera automat sumarul.
- **Acces simultan la înregistrări pentru mai mulți utilizatori.** Cu înregistrările tipărite, dacă două persoane doresc să caute o înregistrare în același timp, a doua persoană trebuie să aștepte ca prima să pună înregistrarea înapoi. MySQL oferă posibilitatea accesului de către mai mulți utilizatori, astfel încât amândoi să poată avea acces simultan la înregistrare.
- **Acces de la distanță la înregistrări și transmiterea electronică a acestora.** Înregistrările tipărite vă impun să vă aflați în punctul de amplasare a acestora, sau trebuie ca o persoană să execute copii ale înregistrărilor și să vi le trimită. Înregistrările electronice creează posibilitatea de acces de la distanță la înregistrări sau posibilitatea de transmitere electronică a acestora. Dacă grupul dumneavoastră de medici stomatologi are asociați în filiale, acești asociați pot avea acces la înregistrările dumneavoastră de la sediile proprii. Nu trebuie să le trimiteți copii prin curier. Dacă o persoană care are nevoie de înregistrări nu dispune de același tip de program de baze de date ca dumneavoastră, dar are un program de poștă electronică, puteți selecta înregistrările dorite și le puteți trimite conținutul pe cale electronică.

Dacă ați mai folosit sisteme de gestiune a bazelor de date, deja sunteți la curent cu avantajele descrise anterior și probabil vă gândiți să depășiți etapa aplicațiilor uzuale de tip „înlocuire a fișelor pe hârtie". Sistemele de baze de date sunt acum folosite pentru a oferi servicii în moduri care nu erau posibile în trecutul apropiat. Modul în care multe instituții folosesc o bază de date în conjuncție cu un sit Web este un bun exemplu.

Să presupunem că firma dumneavoastră are o bază de date inventar care este folosită de personalul de la biroul de service atunci când clienții sună să afle dacă aveți sau nu un articol în stoc, precum și prețul acestuia. Aceasta este o utilizare relativ tradițională a unei baze de date. Totuși, dacă firma dumneavoastră alcătuiește un sit Web pentru a fi vizitat de clienți, puteți oferi un serviciu suplimentar, și anume o pagină de căutare care permite clienților să determine prețul și prezența în stoc a unui produs. Acest lucru oferă clienților informațiile pe care le doresc, iar modul în care le oferiți constă în căutarea automată a informațiilor de inventar stocate în baza de date, pentru a se găsi articolele în chestiune. Clientul obține informația imediat, fără a fi pus „în așteptare" și obligat să asculte o muzică înregistrată agasantă și fără a fi obligat să respecte orele de funcționare a biroului de service. Fiecare client care folosește situl dumneavoastră de Web înseamnă un apel telefonic în minus de rezolvat de către o persoană de la biroul de asistență. (Poate că astfel situl Web își amortizează investiția.)

Puteți utiliza baza de date chiar mai bine. Cererile bazate pe Web de căutare în inventar pot furniza informații nu numai clienților, dar și dumneavoastră. Interogările vă arată ce anume caută clienții, iar rezultatele interogărilor vă arată dacă sunteți sau nu în măsură să le satisfaceți cererile. Dacă nu aveți ceea ce caută clienții, riscați să pierdeți afacerea. Deci, este logic să înregistrați informații despre căutările în inventar, și anume produsele căutate de clienți, respectiv dacă le aveți sau nu în stoc. Apoi, puteți folosi aceste informații pentru a vă ajusta inventarul și pentru a asigura clienților dumneavoastră servicii mai bune.

O altă aplicație recentă pentru bazele de date este deservirea publicității pe manșete în paginile Web. Nici mie nu-mi plac aceste reclame mai mult decât dumneavoastră, dar nu e mai puțin adevărat că reprezintă o aplicație populară pentru MySQL, care poate fi folosit pentru stocarea reclamelor și regăsirea lor în vederea afișării de către un server Web. De asemenea, MySQL poate executa operațiile de evidență a înregistrărilor asociate deseori acestei activități, urmărind reclamele care au fost afișate, numărul de afișări, siturile care au obținut accesul la acestea etc.

Deci, cum funcționează MySQL? Cea mai bună modalitate de a afla este de a încerca personal, iar pentru aceasta vom avea nevoie de o bază de date de lucru.

O bază de date demonstrativă

Această secțiune descrie baza de date demonstrativă pe care o vom folosi pe tot parcursul volumului de față. Vă este oferită o sursă de exemple pe care le puteți încerca pe măsură ce învățați să utilizați MySQL. Vom extrage exemple mai ales din două dintre situațiile prezentate anterior:

- **Scenariul de organizare cu secretarul.** Avem nevoie de ceva mai bine definit decât o „instituție", deci voi crea una cu următoarele caracteristici: este compusă din persoane care au în comun o afinitate pentru istoria Statelor Unite (denumită, în lipsa unui nume mai bun, Liga istorică americană). Membrii își păstrează această calitate prin reînnoirea afilierii, plătind o cotizație. Cotizațiile sunt alocate cheltuielilor Ligii, cum ar fi publicarea unui buletin de știri, denumit „Cronici ale trecutului SUA". De asemenea, Liga dispune de un mic sit Web, dar acesta nu este foarte dezvoltat. Până acum, situl a fost limitat la informații elementare, legate de scopurile Ligii, identitatea persoanelor din conducere și modul de înscriere în organizație.
- **Scenariul de evidență a rezultatelor școlare.** Pe durata perioadei de notare, administrați chestionare și teste, înregistrați punctaje și dați note. După aceea, determinați notele finale, pe care le predați secretariatului școlii alături de o situație a prezenței elevilor.

Acum, să examinăm mai îndeaproape aceste situații din punctul de vedere a două cerințe:

- Trebuie să decideți ce anume doriți să obțineți cu baza de date; cu alte cuvinte, scopurile pe care doriți să le atingeți.
- Trebuie să determinați ce anume veți insera în baza de date; cu alte cuvinte, datele a căror evidență doriți să o păstrați.

Poate părea ciudat să ne gândim la rezultatele unei baze de date înainte de a avea în vedere datele ce trebuie introduse în aceasta. La urma urmelor, trebuie să introduceți date înainte de a le putea regăsi. Dar modul de utilizare a unei baze de date este determinat de scopurile dumneavoastră, iar acestea sunt asociate mai strâns cu rezultatele bazei de date decât cu datele inserate în aceasta. În mod categoric nu doriți să irosiți timp și efort inserând informații într-o bază de date, decât dacă intenționați să le folosiți într-un anume scop în viitor.

Liga istorică americană

Situația inițială a acestui scenariu este aceea că dumneavoastră, ca secretar al Ligii, administrați lista cu membri folosind un document creat cu un procesor de texte. Această situație este adecvată pentru generarea unui catalog tipărit, dar reduce posibilitățile de utilizare a informațiilor în alte moduri. Aveți în vedere următoarele obiective:

- Doriți să puteți produce date de ieșire din catalog în diferite formate, folosind numai informații adecvate aplicației. Un deziderat este posibilitatea de a genera catalogul tipărit în fiecare an – o cerință din trecut a Ligii, pe care dumneavoastră intenționați să continuați a o îndeplini. Vă puteți gândi și la alte utilizări pentru informațiile din catalog – de exemplu, de a furniza lista membrilor actuali pentru programul editat cu ocazia banchetului anual al Ligii. Aceste aplicații implică diferite seturi de informații. Catalogul tipărit folosește întregul conținut al intrării corespunzătoare fiecărui membru. Pentru programul de la banchet, trebuie să extrageți numai numele membrilor (operație deloc ușoară dacă se folosește un procesor de texte).
- Doriți să căutați în catalog pe acei membri ale căror intrări satisfac diferite criterii. De exemplu, doriți să știți care sunt membrii care trebuie să-și plătească în curând cotizația. O altă aplicație care necesită căutări apare din lista de cuvinte cheie aferentă fiecărui membru (de exemplu Războiul de Secesiune, criza din Wall Street, drepturile civile sau viața lui Thomas Jefferson). Uneori, unii membri vă cer o listă a altor membri cu interese similare cu ale lor, iar dumneavoastră doriți să puteți satisface aceste cereri.

- Doriți să plasați catalogul pe Internet, în situl Web al Ligii, ceea ce ar fi avantajos atât pentru membrii Ligii, cât și pentru dumneavoastră. Dacă puteți converti catalogul în pagini Web printr-un proces oarecare cu un grad adecvat de automatizare, o versiune on-line a catalogului poate fi menținută „la zi" într-o manieră mai oportună decât versiunea tipărită. Iar dacă în acest catalog electronic se pot efectua căutări, membrii pot căuta personal informații, cu ușurință. De exemplu, un membru care dorește să știe care sunt membrii interesați de Războiul de Secesiune poate afla această informație fără a fi nevoit să vă aștepte să efectuați căutarea, iar dumneavoastră nu veți mai avea nevoie să găsiți timp pentru a efectua personal căutarea.

Sunt perfect conștient de faptul că o bază de date nu este cel mai interesant lucru din lume, deci n-am de gând să fac cine știe ce afirmații trăsnite, cum că o bază de date stimulează gândirea creatoare. Totuși, dacă încetați să asimilați informațiile ca fiind o entitate cu care trebuie să luptați (așa cum faceți cu documentul dumneavoastră creat cu procesorul de texte) și începeți să le asimilați cu o entitate pe care o puteți manipula relativ ușor (așa cum sperați să procedați cu SQL), acest lucru va avea un efect eliberator asupra capacității dumneavoastră de a veni cu noi modalități de a folosi sau de a prezenta acele informații:

- Dacă informațiile din baza de date pot fi mutate în situl Web sub forma unui catalog electronic, puteți face ca informațiile să se „scurgă" și în sens invers. De exemplu, dacă membrii își pot edita propriile rubrici on-line pentru a actualiza baza de date, nu mai este necesar să efectuați personal toate modificările și astfel s-ar contribui la sporirea acurateței informațiilor din catalog.
- Dacă ați stocat adrese de e-mail în baza de date, le puteți folosi pentru a trimite mesaje e-mail membrilor care nu și-au mai actualizat rubricile în ultimul timp. Mesajele pot indica membrilor conținutul curent al rubricii lor, le pot cere să-l reînnoiască și pot indica modalitatea de efectuare a diferitelor modificări, folosind facilitățile furnizate pe situl Web.
- O bază de date poate contribui la sporirea utilității unui sit Web în moduri care nu au nici cea mai mică legătură cu lista membrilor. Liga publică un buletin de știri, „Cronici ale trecutului SUA", care are în fiecare număr o secțiune dedicată copiilor, ce conține un chestionar pe teme istorice. Unele dintre numerele recente s-au concentrat asupra unor date biografice ale președinților americani. Situl Web poate include de asemenea o secțiune pentru copii, unde chestionarele sunt puse on-line. Această secțiune poate căpăta, eventual, un caracter interactiv, prin inserarea în baza de date a informațiilor care au stat la baza chestionarelor și determinând serverul Web să interogheze baza de date în căutarea unor întrebări care vor fi prezentate aleator.

Bun! În acest moment, numărul de utilizări pentru baza de date la care v-ați gândit vă poate duce la concluzia că v-ați cam lăsat dus de val. După o scurtă pauză pentru a reveni din nou pe Pământ, începeți să vă puneți unele întrebări practice:

- **Nu e puțin cam prea ambițios?** Nu e cam multă muncă pentru toate astea? Orice pare mai ușor atunci când doar vă gândiți și nu puneți în practică, desigur, iar eu nu mă voi preface spunând că toate acestea sunt banal de implementat. Totuși, la finele acestei cărți veți fi realizat deja tot ce am amintit anterior. Rețineți un singur lucru: nu este necesar să faceți totul deodată. Vom împărți sarcina în componente și vom aborda fiecare componentă în parte.

- **Poate face MySQL toate astea?** Nu, nu poate. De exemplu, MySQL nu dispune de caracteristici directe pentru interacțiunea cu Web-ul. Totuși, chiar dacă MySQL nu poate face singur tot ceea ce am discutat, există alte instrumente care pot colabora cu MySQL pentru a-i completa și extinde posibilitățile.

  Vom folosi limbajul de scripting Perl și modulul Perl DBI (interfață cu bazele de date) pentru a scrie scripturi care obțin acces la bazele de date MySQL. Perl dispune de excelente posibilități de prelucrare a textelor, care permit manipularea rezultatelor unei interogări într-o manieră extrem de flexibilă, pentru a obține date de ieșire într-o varietate de formate. De exemplu, putem folosi Perl pentru a genera catalogul în format Rich Text Format (RTF), un format care poate fi citit de către toate categoriile de procesoare de texte.

  De asemenea vom folosi PHP, un alt limbaj de scripting. PHP este adaptat pentru scrierea aplicațiilor Web și lucrează cu bazele de date. Acesta vă permite să rulați interogări MySQL direct din paginile Web și să generați pagini noi, care includ rezultatele interogărilor bazelor de date. PHP colaborează bine cu Apache (cel mai popular server Web din lume), simplificând efectuarea unor operații precum prezentarea unui formular de căutare și afișarea rezultatelor căutării.

  MySQL se integrează bine cu aceste instrumente și vă oferă flexibilitatea de a le combina în funcție de preferințele dumneavoastră, optând pentru realizarea scopurilor scontate.

  Nu sunteți obligat să folosiți componentele unui singur set de programe cu posibilități de integrare clamate pe toate tonurile, dar care nu pot funcționa bine decât unele cu altele.
- **În final, marea întrebare: cât vor costa toate acestea?** La urma urmelor, Liga are un buget limitat. S-ar putea să vă surprindă, dar probabil că nu va costa nimic. Dacă sunteți un cunoscător al sistemelor de baze de date, atunci știți că sunt destul de costisitoare. Prin contrast, MySQL este de regulă gratuit. Există unele situații când aveți nevoie de o licență, dar aceasta costă numai 200 de dolari pentru un număr nelimitat de utilizatori. (Vezi prefața pentru îndrumări generale privind licența, precum și manualul de referință MySQL pentru detalii concrete.) Celelalte instrumente pe care le vom folosi (Perl, DBI, PHP, Apache) sunt gratuite deci, dacă ținem cont de toate aspectele, puteți alcătui un sistem util fără nici o cheltuială.

Alegerea sistemului de operare pentru dezvoltarea bazei de date vă aparține. Toate programele pe care le vom discuta rulează sub UNIX, iar majoritatea lor rulează sub Windows. Eu vă recomand să folosiți MySQL și celelalte instrumente sub UNIX. Toate instrumentele își au originea în UNIX și de acolo au fost portate ulterior spre Windows, fiind desfășurate mai frecvent în UNIX decât în Windows. Aceasta înseamnă că versiunile Windows au avut o perioadă de maturizare mai scurtă și că nu au fost testate și utilizate atât de aprofundat ca versiunile sub UNIX.

Acum, să abordăm și cealaltă situație pentru care vom folosi baza de date demonstrativă.

Proiectul de evidență a rezultatelor școlare

Scenariul inițial în acest caz este acela că, în calitate de profesor, aveți și responsabilități legate de evidența rezultatelor școlare. Doriți să convertiți procesul de calcul al notelor de la o operație manuală (folosind un catalog școlar) la o reprezentare electronică, folosind MySQL. În acest caz, rezultatele pe care doriți să le obțineți cu baza de date sunt implicite, prin modul în care folosiți catalogul în prezent:

- Pentru fiecare chestionar sau test, înregistrați punctajele. Pentru teste, puneți punctajele în ordine, astfel încât să le puteți examina și determina ajustările pentru fiecare notă literală (A, B, C, D, E și F).
- La finalul perioadei de notare, calculați punctajul total al fiecărui elev, apoi sortați rezultatele totale și determinați notele în funcție de acestea. Rezultatele totale pot necesita calcule cu ponderi, deoarece doriți să acordați testelor o importanță mai mare decât chestionarelor.
- La sfârșitul perioadei de notare, furnizați secretariatului școlii informații despre prezența elevilor la curs.

Obiectivele sunt de a evita sortarea manuală și totalizarea manuală a punctajelor și a evidențelor privind prezența. Cu alte cuvinte, doriți ca MySQL să sorteze punctajele și să efectueze calculele necesare pentru a determina punctajul total al fiecărui elev și numărul de absențe la încheierea perioadei de notare. Pentru a realiza aceste deziderate, veți avea nevoie de lista elevilor din clasă, de punctajele pentru fiecare chestionar și test, precum și datele la care elevul a lipsit.

Cum se poate aplica baza de date demonstrativă în cazul dumneavoastră

Dacă nu vă interesează în mod deosebit Liga istorică sau evidența rezultatelor școlare, vă puteți întreba ce legătură au toate acestea cu dumneavoastră. Răspunsul este că aceste scenarii-exemplu nu sunt un scop în sine. Acestea nu fac decât să furnizeze un vehicul prin care să se poată ilustra operațiile posibile cu MySQL și cu instrumentele asociate acestuia.

Cu puțină imaginație, veți vedea cum exemplele de interogări ale bazelor de date se aplică problemelor specifice pe care doriți să le rezolvați. Să presupunem că lucrați în acel cabinet stomatologic de care am vorbit anterior. Nu veți vedea în această carte prea multe interogări din domeniul stomatologic, dar veți observa că multe dintre interogări se pot aplica propriilor dumneavoastră interese. De exemplu, determinarea acelor membri ai Ligii istorice care trebuie să-și achite în curând cotizațiile este similară cu determinarea pacienților care n-au mai trecut de mult pe la dentist. Ambele sunt interogări bazate pe date deci, o dată ce ați învățat să scrieți interogarea de determinare a persoanelor ce au de plătit cotizația, o puteți aplica pentru a scrie interogarea de depistare a pacienților „delincvenți", care vă interesează mai mult.

O terminologie esențială a bazelor de date

Poate ați observat că ați citit deja mai multe pagini dintr-o carte despre baze de date și încă nu ați dat peste terminologia și jargonul de specialitate. De fapt, încă nu am spus nimic despre modul în care arată efectiv o „bază de date", chiar dacă avem o idee brută cu privire la modul în care va fi folosită baza noastră de date demonstrativă. Totuși, suntem pe cale să proiectăm acea bază de date, după care vom începe să o implementăm, deci nu mai putem evita terminologia. Acesta este și scopul secțiunii de față: să descrie unii termeni care apar pe parcursul cărții, pentru ca dumneavoastră să-i cunoașteți. Din fericire, multe concepte legate de bazele de date relaționale sunt, în realitate, foarte simple. De fapt, o bună parte din atractivitatea bazelor de date provine din simplitatea conceptelor lor fundamentale.

Terminologie structurală

În lumea bazelor de date, MySQL este clasificat ca *sistem de gestiune a bazelor de date relaționale* (SGBDR). Această expresie poate fi divizată după cum urmează:

- Baza de date (literele BD din abrevierea SGBDR) constituie depozitul pentru informațiile pe care doriți să le stocați, structurat într-un mod simplu, obișnuit:
  - Colecția de date dintr-o bază de date este organizată în tabele.
  - Fiecare tabel este organizat în rânduri și coloane.
  - Fiecare rând dintr-un tabel este considerat o înregistrare.
  - Înregistrările pot conține mai multe componente de informație; fiecare coloană dintr-un tabel corespunde uneia din aceste componente.
- Sistemul de gestiune (literele SG) este programul care vă permite să vă folosiți datele prin inserarea, regăsirea, modificarea sau ștergerea înregistrărilor.
- Cuvântul „relațional" (inițiala R) indică un anumit tip de SGBD, adică unul care se pricepe foarte bine la relaționarea (în speță coroborarea) informațiilor stocate într-un tabel cu informațiile stocate într-un alt tabel, prin căutarea elementelor comune din fiecare tabel. Puterea unui SGBD relațional constă în capacitatea sa de a extrage în mod convenabil date din aceste tabele și de a combina informații din tabele corelate pentru a genera răspunsuri la întrebări la care nu se poate răspunde numai pe baza tabelelor individuale.

Iată un exemplu care prezintă modul în care o bază de date relațională organizează datele în tabele și corelează informațiile dintr-un tabel cu un alt tabel. Să presupunem că administrați un sit Web care include un serviciu de publicitate pe manșete. Stabiliți contracte cu companiile care doresc ca reclamele lor să fie afișate atunci când sunt vizitate paginile din situl dumneavoastră. De fiecare dată când un vizitator deschide una din paginile dumneavoastră, furnizați reclama înglobată în pagină browserului vizitatorului și percepeți companiei o mică taxă. Pentru a reprezenta aceste informații, folosiți trei tabele (vezi figura 1.1). Un tabel, `company` (companie) are coloane pentru numele companiei, numărul, adresa și numărul de telefon. Un alt tabel, `ad` (reclamă), prezintă numerele reclamelor, numărul companiei care „deține" reclama și tariful pe care îl percepeți

pentru fiecare deschidere a paginii. Cel de-al treilea tabel, `hits` (reușite) conține evidența citirii reclamelor în funcție de numărul reclamei și de data la care a fost furnizată reclama.

company table

company_name	company_num	address	phone
Big deal, Ltd.	13	14 Grand Blvd.	875-2934
Pickles, Inc.	14	59 Cucumber Dr.	884-2472
Real Roofing Co.	17	928 Shingles Rd.	882-4173
GigaFred & Son	23	2572 Family Ave.	847-4738

ad table

company_num	ad_num	hit_fee
14	48	0.01
23	49	0.02
17	52	0.01
13	55	0.03
23	62	0.02
23	63	0.01
23	64	0.02
13	77	0.03
23	99	0.03
14	101	0.01
13	102	0.01
17	119	0.02

hit table

ad_num	date
49	July 13
55	July 13
48	July 14
63	July 14
101	July 14
62	July 14
119	July 14
102	July 14
52	July 14
48	July 14
64	July 14
119	July 14
48	July 14
101	July 14
63	July 15
49	July 15
77	July 15
99	July 15

Figura 1.1 Tabele pentru evidența reclamelor pe manșete.

Din aceste informații, se poate răspunde la unele întrebări folosind un singur tabel. Pentru a determina numărul de companii cu care aveți contracte, trebuie numai să numărați rândurile din tabelul `company`. Similar, pentru a determina numărul de reușite (citiri) într-o anumită perioadă de timp, este necesară numai examinarea tabelului `hit`. Alte întrebări sunt mai complexe și este necesară consultarea mai multor tabele pentru a determina răspunsurile. De exemplu, pentru a determina de câte ori a fost servită fiecare reclamă a companiei Pickles, Inc. pe data de 14 iulie, veți folosi toate cele trei tabele, după cum urmează:

1. Căutați numele companiei (Pickles, Inc.) în tabelul `company` pentru a determina numărul companiei.
2. Folosiți numărul companiei pentru a descoperi înregistrările corespunzătoare în tabelul `ad` pentru a determina numerele asociate ale reclamelor. Există două asemenea reclame, în speță 48 și 101.
3. Pentru fiecare dintre înregistrările echivalente din tabelul `ad`, folosiți numărul reclamei din înregistrare pentru a descoperi înregistrările corespunzătoare din tabelul `hit` care se încadrează la data respectivă, după care numărați echivalențele. Există trei echivalențe pentru reclama 48 și două echivalențe pentru reclama 101.

Pare complicat, dar acesta este chiar domeniul în care excelează sistemele de baze de date relaționale. Complexitatea este de fapt oarecum iluzorie, deoarece fiecare din etapele descrise anterior reprezintă ceva mai mult decât o simplă operație de echivalare: stabiliți o corelație între un tabel și alt tabel prin echivalențe între valorile din rândurile unui tabel cu valorile din rândurile unui alt tabel. Aceeași operație simplă poate fi exploatată în diferite moduri, pentru a se răspunde la toate categoriile de întrebări: Câte reclame diferite are fiecare companie? Care este compania cu reclamele cele mai populare? Care este venitul pe care îl aduce fiecare reclamă? Care este onorariul total ce trebuie plătit de fiecare companie pentru perioada de tarifare curentă?

Acum știți suficientă teorie privind bazele de date relaționale pentru a înțelege restul cărții, motiv pentru care nu trebuie să vorbim despre a treia formă normală, diagrame entitate-relație și alte chestii de acest gen. Dacă chiar doriți să aflați detalii despre aceste lucruri, este excelent, dar ați greșit cartea. Vă propun să începeți prin a consulta autori ca C.J. Date sau E.F. Codd.

Terminologia limbajului de interogare

Pentru a comunica cu MySQL, folosiți un limbaj denumit SQL (Structured Query Language – limbaj de interogare structurat). SQL este în prezent limbajul standard pentru baze de date și este acceptat de toate sistemele importante de baze de date. SQL include numeroase categorii diferite de instrucțiuni, toate fiind concepute pentru a vă permite să interacționați cu baza dumneavoastră de date în moduri interesante și utile.

Ca orice alt limbaj, SQL poate părea ciudat atunci când îl învățați pentru prima dată. De exemplu, pentru a crea un tabel, trebuie să indicați sistemului MySQL care va fi structura tabelului. Dumneavoastră și cu mine ne gândim la un tabel ca la o diagramă sau o imagine, dar MySQL nu, deci veți crea tabelul indicând sistemului MySQL ceva de genul acesta:

```
CREATE TABLE companie
(
    nume_companie CHAR(30),
    nr_companie INT,
    adresa CHAR(30),
    telefon CHAR(12)
)
```

Dacă sunteţi începător în domeniul SQL, instrucţiuni ca acestea pot părea oarecum impunătoare, dar nu trebuie să fiţi programator pentru a învăţa să folosiţi SQL în mod eficient. Pe măsură ce vă familiarizaţi cu limbajul, veţi privi instrucţiunea `CREATE TABLE` într-o altă lumină – ca pe un aliat care vă ajută să vă descrieţi informaţiile, nu ca pe o bălmăjeală ciudată.

Terminologia folosită în arhitectura sistemului MySQL

Când folosiţi MySQL, în realitate utilizaţi două programe, deoarece MySQL operează folosind o arhitectură client/server:

- Serverul de baze de date este un program localizat în calculatorul unde sunt stocate datele dumneavoastră. Acesta ascultă cererile clienţilor care sosesc prin reţea şi obţine acces la conţinutul bazei de date în funcţie de aceste cereri, pentru a furniza clienţilor informaţiile pe care le solicită.
- Clienţii sunt programe care se conectează la serverul de baze de date şi emit interogări pentru a-i indica acestuia informaţiile pe care le doresc.

Distribuţia MySQL include serverul şi numeroase programe client. Clienţii sunt utilizaţi în funcţie de scopurile urmărite. Cel mai folosit program este `mysql`, un client interactiv care vă permite să emiteţi interogări şi să vedeţi rezultatele. Alţi clienţi sunt `mysqldump` şi `mysqlimport`, care trimit conţinutul tabelelor într-un fişier şi invers, respectiv `mysqladmin`, care vă permite să verificaţi starea serverului şi execută operaţii administrative, cum ar fi solicitarea de oprire a serverului. Dacă aveţi aplicaţii pentru care utilizarea clienţilor standard nu este adecvată, MySQL mai furnizează o bibliotecă de programare a clienţilor, astfel încât dumneavoastră să vă puteţi scrie propriile programe. Biblioteca este utilizabilă direct din programele C, după cum există şi numeroase alte interfeţe, dacă preferaţi alt limbaj decât C.

Arhitectura client/server din MySQL are anumite avantaje:

- **Serverul asigură controlul concurenţei, pentru ca doi utilizatori să nu poată modifica aceeaşi înregistrare simultan.** Toate cererile clienţilor trec prin server, deci serverul determină clientul care execută cutare operaţie şi când o execută. Dacă mai mulţi clienţi doresc simultan accesul la acelaşi tabel, ei nu sunt obligaţi să se găsească şi să negocieze unii cu alţii. Ei îşi trimit cererile la server şi îl lasă pe acesta să se ocupe de determinarea ordinii în care vor fi onorate cererile.
- **Nu trebuie să deschideţi sesiunea de lucru la calculatorul unde este conectată baza dumneavoastră de date.** MySQL „ştie" să lucreze prin Internet, deci puteţi rula un program client de oriunde vă aflaţi, iar clientul se poate conecta la server prin reţea. Distanţa nu contează; puteţi obţine acces la server din orice parte a lumii. Dacă serverul se află pe un calculator din Australia, vă puteţi lua calculatorul laptop într-o călătorie în Islanda şi tot puteţi obţine acces la baza dumneavoastră de date.

Aceasta înseamnă oare că *oricine* poate avea acces la datele dumneavoastră, prin Internet? Nu. MySQL include un sistem flexibil de securitate, deci puteţi permite accesul numai persoanelor îndrituite să beneficieze de el. De asemenea, vă puteţi asigura că aceste persoane pot face numai ceea ce trebuie să facă. Poate că Sally de la biroul de facturare trebuie să poată citi şi actualiza (modifica) înregistrările, în timp ce Phil de la biroul de service trebuie să poată numai să citească înregistrările. Le puteţi stabili privilegiile în consecinţă. Dacă doriţi să rulaţi un sistem autonom, pur şi simplu stabiliţi privilegiile de acces astfel încât clienţii să se poată conecta numai de la gazda pe care rulează serverul.

Un manual MySQL

Acum, aveţi toate cunoştinţele necesare; este momentul să puneţi MySQL la lucru!

Această secţiune vă va ajuta să vă familiarizaţi cu MySQL, propunându-vă un manual pe care să-l încercaţi. În timp ce parcurgeţi manualul, veţi crea o bază de date demonstrativă şi tabelele care o compun, apoi veţi interacţiona cu baza de date prin adăugarea, regăsirea, ştergerea şi modificarea informaţiilor. În plus, pe durata lucrului cu baza de date demonstrativă, veţi învăţa următoarele aspecte:

- Cum să comunicaţi cu un server MySQL folosind programul client `mysql`.
- Elementele fundamentale ale limbajului SQL pe care MySQL le înţelege. (În cazul în care cunoaşteţi deja SQL din utilizarea unui alt SGBDR, este o idee bună să parcurgeţi rapid acest manual, pentru a vedea dacă versiunea SQL a sistemului MySQL diferă de versiunea pe care o cunoaşteţi.)

Aşa cum s-a arătat în secţiunea anterioară, MySQL operează folosind o arhitectură client/server, în cadrul căreia serverul rulează pe calculatorul care conţine bazele de date, iar clienţii se conectează la server printr-o reţea. Acest manual se bazează în cea mai mare parte pe utilizarea programului client `mysql`. Clientul citeşte interogările SQL emise de dumneavoastră, le trimite serverului pentru a fi executate şi afişează rezultatele, pentru ca dumneavoastră să puteţi vedea ce s-a întâmplat. `mysql` rulează pe toate platformele acceptate de MySQL şi constituie metoda cea mai directă de a interacţiona cu serverul, deci este logic să începem cu acest client.

Pe parcursul acestei cărţi, vom folosi `samp_db` ca nume al bazei noastre de date demonstrative. Uneori, va fi necesar să folosiţi un alt nume pentru baza de date în timp ce parcurgeţi exemplele. Poate că altă persoană din sistemul dumneavoastră foloseşte deja numele `samp_db` sau poate că administratorul v-a repartizat un alt nume pentru baza de date. În ambele cazuri, înlocuiţi `samp_db` cu numele real al bazei dumneavoastră de date ori de câte ori vedeţi primul nume în exemple.

Numele tabelelor pot fi folosite exact aşa cum apar în exemple, chiar dacă mai multe persoane din sistemul dumneavoastră îşi au propriile lor baze de date demonstrative. În MySQL, nu contează dacă o altă persoană foloseşte aceleaşi nume de tabele. Atâta timp cât fiecare din dumneavoastră îşi are propria sa bază de date, MySQL va păstra numele nealterate şi va împiedica interferenţele între utilizatori.

Cerinţe preliminare

Pentru a încerca exemplele din acest manual, este necesar ca MySQL să fie instalat. În particular, trebuie să aveţi acces la clienţii MySQL şi la un server MySQL oarecare. Programele client trebuie să fie instalate în calculatorul dumneavoastră. Veţi avea nevoie cel puţin de `mysql`, iar `mysqlimport` este de asemenea util. Serverul se poate afla în cal-

culatorul dumneavoastră, deși acest lucru nu este necesar. Dacă aveți permisiunea să vă conectați la server, acesta poate fi amplasat oriunde.

Dacă se întâmplă ca serverul să ruleze pe calculatorul dumneavoastră, programele client sunt probabil de asemenea instalate, iar dumneavoastră sunteți pregătit. Dacă trebuie să vă procurați MySQL, vezi Anexa A, „Obținerea și instalarea programelor", pentru instrucțiuni. Fie consultați capitolul respectiv (dacă instalați personal MySQL), fie arătați-l administratorului dumneavoastră de sistem. Dacă accesul dumneavoastră la rețea are loc prin intermediul unui furnizor de servicii Internet (ISP), verificați dacă furnizorul dumneavoastră dispune sau nu de MySQL. Dacă furnizorul dumneavoastră nu oferă serviciul MySQL și nu este dispus să-l instaleze, consultați Anexa J, „Furnizori de servicii Internet", pentru sfaturi cu privire la alegerea unui furnizor mai adecvat.

În afara programului MySQL, aveți nevoie de permisiunea de a crea baza de date demonstrativă și tabelele sale. Dacă nu aveți permisiunea, cereți-o administratorului dumneavoastră MySQL. Administratorul vă poate acorda această permisiune rulând clientul `mysql` și emițând comenzi ca acestea:

```
GRANT ALL ON samp_db.* TO paul@localhost IDENTIFIED BY "secret"
GRANT ALL ON samp_db.* TO paul@% IDENTIFIED BY "secret"
```

Diferența între MySQL și `mysql`

Pentru a evita confuziile, trebuie să precizăm că "MySQL" se referă la întregul SGBDR MySQL, iar "`mysql`" este numele unui anumit program client. Cele două se pronunță la fel, dar le vom deosebi prin diferențele determinate prin majuscule și tipurile de literă.

Prima comandă oferă utilizatorului `paul` un acces complet la baza de date `samp_db` și la toate tabelele acesteia, atunci când `paul` se conectează de la calculatorul `localhost` (aceeași gazdă pe care rulează serverul). De asemenea, atribuie parola `secret`. A doua comandă este similară, dar permite lui `paul` să se conecteze de la orice gazdă (`%` este un caracter de înlocuire). De asemenea, puteți înlocui `%` cu un anumit nume de gazdă, pentru a permite lui `paul` să se conecteze chiar de la gazda respectivă. (O atare instrucțiune `GRANT` poate fi necesară dacă serverul dumneavoastră permite acces anonim de la `localhost`, datorită modului în care serverul caută în tabelele de acordare echivalențe cu conexiunile de intrare.) Mai multe informații despre instrucțiunea `GRANT` și despre configurarea conturilor de utilizator MySQL pot fi găsite în capitolul 11, „Administrarea generală a sistemului MySQL".

Obținerea distribuției bazei de date demonstrative

Acest manual face referire, în anumite situații, la fișiere din „distribuția bazei de date demonstrative". Acestea sunt fișiere care conțin interogări sau date care vă vor ajuta să configurați baza de date demonstrativă. Pentru a vă procura această distribuție, vezi Anexa A. Când despachetați distribuția, aceasta va crea un catalog denumit `samp_db`, care conține fișierele de care aveți nevoie. Vă sugerez să comutați în catalogul respectiv ori de câte ori încercați exemple referitoare la baza de date demonstrativă.

Stabilirea și întreruperea conexiunilor cu serverul

Pentru a vă conecta la serverul dumneavoastră, invocați programul `mysql` din interpretorul dumneavoastră (adică de la promptul UNIX sau de la o consolă DOS sub Windows). Comanda arată astfel:

```
% mysql optiuni
```

Am folosit caracterul `%` în carte pentru a indica promptul interpretorului. Acesta este unul din prompturile UNIX standard; un altul este `$`. În Windows, promptul este ceva de genul `C:\>`.

Partea `optiuni` a liniei de comandă `mysql` poate fi vidă, dar cel mai probabil va trebui să emiteți o comandă ca următoarea:

```
% mysql -h nume_gazda -u nume_utilizator -p
```

Este posibil să nu fie necesar să specificați toate aceste opțiuni când invocați `mysql`; apelați la administratorul MySQL pentru a afla sintaxa exactă a comenzii pe care trebuie să o folosiți. Probabil că va trebui să specificați cel puțin un nume și o parolă.

Când sunteți la primii pași în învățarea sistemului MySQL, probabil că sistemul său de securitate vi se va părea agasant, deoarece îngreunează efectuarea operațiilor dorite. (Trebuie să obțineți permisiunea de a crea și de a obține acces la o bază de date și trebuie să vă specificați numele și parola de fiecare dată când vă conectați la server.) Totuși, după ce veți trece dincolo de baza de date demonstrativă și veți introduce și folosi propriile dumneavoastră înregistrări, perspectiva dumneavoastră se va schimba radical. Apoi, veți aprecia faptul că MySQL împiedică alte persoane să vă spioneze (sau, mai rău, chiar să vă distrugă!) datele.

Iată care este semnificația opțiunilor:

- `-h` *`nume_gazda`* (formă alternativă: `--host=`*`nume_gazda`*)

  Gazda serverului la care doriți să vă conectați. Dacă serverul rulează pe același calculator pe care utilizați `mysql`, în mod normal această opțiune poate fi omisă.

- `-u` *`nume_utilizator`* (formă alternativă: `--user=`*`nume_utilizator`*)

  Numele dumneavoastră de utilizator MySQL. Dacă folosiți UNIX și numele dumneavoastră de utilizator MySQL este același ca numele dumneavoastră de deschidere a sesiunii de lucru, puteți omite această opțiune; `mysql` va folosi numele de deschidere a sesiunii de lucru ca nume MySQL.

  Sub Windows, numele de utilizator prestabilit este `ODBC`, ceea ce nu este foarte util. Fie specificați un nume în linia de comandă, fie stabiliți un nume prestabilit în mediul dumneavoastră prin configurarea variabilei `USER`. De exemplu, pot folosi următoarea comandă `set` pentru a specifica numele de utilizator `paul`:

  ```
  set USER=paul
  ```

- `-p` (formă alternativă: `--password`)

  Această opțiune indică programului `mysql` să vă solicite parola dumneavoastră MySQL. Notă: vă puteți insera parola în linia de comandă tastând-o sub forma `-p`*`parola_dvs`* (formă alternativă: `--parola=`*`parola_dvs`*). Totuși, din motive de securitate, cel mai bine

este să nu procedaţi astfel. Dacă specificaţi opţiunea -p fără nici o parolă, indicaţi programului mysql să vă solicite o parolă la deschidere. De exemplu:

```
% mysql -h nume_gazda -u nume_utilizator -p
Enter password:
```

Când vedeţi mesajul Enter password: (introduceţi parola), tastaţi parola dumneavoastră. (Parola nu va fi reflectată pe ecran, în caz că vă spionează careva.) Reţineţi că parola dumneavoastră MySQL nu este obligatoriu aceeaşi ca parola UNIX sau Windows.

Dacă omiteţi opţiunea -p în totalitate, mysql presupune că nu aveţi nevoie de ea şi nu o solicită.

Notă: Opţiunile -h şi -u sunt asociate cuvintelor care urmează după acestea, indiferent dacă între opţiune şi cuvântul următor există sau nu un spaţiu. Acest lucru *nu* este valabil pentru opţiunea -p. Dacă specificaţi parola în linia de comandă, nu trebuie să existe nici un spaţiu între -p şi parolă.

De exemplu, să presupunem că numele meu de utilizator MySQL şi parola mea sunt respectiv paul şi secret, şi că doresc să mă conectez la serverul care rulează pe acelaşi calculator la care am deschis sesiunea de lucru. Comanda mysql pentru această operaţie este:

```
% mysql -u paul -p
Enter password: ******
```

După ce am introdus comanda, mysql afişează textul Enter password: pentru a-mi cere parola, iar eu o tastez (şirul ****** arată locul unde am scris secret).

Dacă totul merge bine, mysql afişează un mesaj de salut şi un prompt mysql> prin care îmi arată că aşteaptă să scriu comenzi. Secvenţa de pornire completă se prezintă astfel:

```
% mysql -u paul -p
Enter password: ******
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1805 to server version: 3.22.25-log

Type 'help' for help.

mysql>
```

Pentru a mă conecta la un server care rulează pe un alt calculator, trebuie să specific numele gazdei folosind opţiunea -h. Dacă gazda respectivă are numele pit-viper.snake.net, comanda arată astfel:

```
% mysql -h pit-viper.snake.net -u paul -p
```

În majoritatea exemplelor care urmează şi care prezintă o linie de comandă mysql, voi omite opţiunile -h, -u şi -p, din motive de concizie. Se presupune că veţi introduce opţiunile necesare.

Există modalităţi de a vă configura contul astfel încât să nu fie necesar să tastaţi parametrii de conexiune de fiecare dată când rulaţi mysql. Acest lucru este discutat în secţiunea „Sugestii pentru interacţiunea cu mysql". Puteţi trece chiar acum la secţiunea respectivă, pentru a vă face o idee cu privire la unele din posibilităţile de a simplifica conectarea la server.

După ce stabiliţi o conexiune cu serverul, puteţi încheia sesiunea oricând tastând QUIT:

```
mysql> QUIT
Bye
```

De asemenea, puteţi termina sesiunea tastând Control-D, cel puţin în UNIX.

Emiterea de interogări

După ce v-aţi conectat la server, sunteţi gata de a emite interogări. Această secţiune descrie unele aspecte generale pe care trebuie să le cunoaşteţi despre interacţiunea cu mysql. .

Pentru a introduce o interogare în mysql, pur şi simplu tastaţi-o. La sfârşitul interogării, tastaţi un caracter punct şi virgulă (;) şi apăsaţi pe tasta Enter. Caracterul punct şi virgulă indică programului mysql că interogarea este completă. (De asemenea, puteţi folosi \g pentru terminarea interogărilor, dacă preferaţi să scrieţi două caractere în loc de unul.)

După ce aţi introdus o interogare, mysql o trimite serverului pentru a fi executată. Serverul prelucrează interogarea şi trimite rezultatele înapoi programului mysql, care afişează rezultatul pentru dumneavoastră.

Iată un exemplu de interogare simplă şi rezultatul acesteia:

```
mysql> SELECT NOW()
```

NOW()
1999-07-24 11:02:36

```
1 row in set (0.00 sec)
```

Această comandă indică data şi ora curentă. (Funcţia NOW() nu este utilă ca atare, dar se poate folosi în expresii – de exemplu, pentru a calcula diferenţa dintre data curentă şi o altă dată.)

De asemenea, mysql afişează numărul rândurilor rezultatului. De regulă, nu voi prezenta acest număr în cadrul exemplelor.

Deoarece mysql aşteaptă caracterul punct şi virgulă înainte de a trimite interogarea serverului, nu trebuie să introduceţi toată interogarea pe un singur rând. Puteţi extinde o interogare pe mai multe linii, dacă doriţi:

```
mysql> SELECT NOW(),
    -> USER(),
    -> VERSION()
    -> ;
```

NOW()	USER()	VERSION()
1999-07-24 11:06:16	paul@localhost	3.23.1-alpha-log

Observaţi modul în care promptul se modifică din mysql în după ce aţi introdus prima linie a interogării; acest lucru vă arată că mysql crede că încă mai introduceţi interogarea. Aceasta este o reacţie importantă, deoarece, dacă uitaţi să plasaţi caracterul punct şi virgulă la sfârşitul unei interogări, promptul vă ajută să sesizaţi că mysql continuă să aştepte ceva. În caz contrar, dumneavoastră veţi aştepta, întrebându-vă de ce o avea nevoie MySQL de atâta timp pentru a vă executa interogarea, iar mysql va aştepta la rândul lui, întrebându-se de ce aveţi nevoie de atâta timp pentru a termina de introdus interogarea!

În majoritatea cazurilor, nu are importanță dacă introduceți interogările folosind majuscule, minuscule sau litere de diferite mărimi. Toate aceste interogări sunt echivalente:

```
SELECT USER()
select user()
SeLeCt UsEr()
```

Exemplele din această carte folosesc majuscule pentru cuvintele cheie și numele de funcții din SQL, respectiv minuscule pentru nume de baze de date, tabele și coloane.

Când invocați o funcție într-o interogare, nu trebuie să existe spații între numele funcției și paranteza care urmează după acesta:

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 1999-07-17 12:44:52 |
+---------------------+
mysql> SELECT NOW ();
ERROR 1064: You have an error in your SQL syntax near '()' at line 1
```

(Aveți o eroare în sintaxa SQL lângă '( )' pe linia 1)

Cele două interogări par similare, dar a doua eșuează, deoarece paranteza nu apare imediat după numele funcției.

Dacă ați început să scrieți o interogare cu mai multe linii și decideți că nu doriți să o executați, tastați `\c` pentru a o anula (cancel):

```
mysql> SELECT NOW(),
    -> VERSION(),
    -> \c
mysql>
```

Remarcați că promptul revine la forma `mysql>`, indicând că `mysql` este gata pentru o nouă interogare.

Puteți stoca interogările într-un fișier și puteți cere programului `mysql` să citească interogările din fișier, în loc să aștepte ca dumneavoastră să le introduceți de la tastatură. Folosiți pentru aceasta facilitățile interpretorului de redirecționare a datelor de intrare. De exemplu, dacă am interogări stocate într-un fișier denumit `fisierul_meu.sql`, pot executa conținutul fișierului astfel:

```
% mysql < fisierul_meu.sql
```

Puteți denumi fișierul cum doriți. Eu folosesc sufixul `sql` pentru a arăta că fișierul conține instrucțiuni SQL.

Execuția programului `mysql` în acest mod va reapărea în secțiunea „Adăugarea de înregistrări noi" atunci când introducem date în baza de date `samp_db`. Este mult mai comod să încărcăm un tabel determinând programul `mysql` să citească instrucțiuni `INSERT` dintr-un fișier decât tastând manual fiecare instrucțiune.

În continuare, acest manual va prezenta multe interogări pe care le puteți încerca pe cont propriu. Acestea sunt indicate prin promptul `mysql>` care precede interogarea și prin caracterul punct și virgulă (de terminare) plasat după aceasta, iar asemenea exemple sunt de regulă însoțite de rezultatul interogării. Trebuie să puteți tasta aceste interogări așa cum sunt prezentate, iar datele de ieșire rezultate trebuie să fie identice.

Interogările prezentate fără un prompt sau fără caracterul punct și virgulă de terminare a instrucțiunii sunt menite pur și simplu a ilustra o idee și nu trebuie executate. (Le puteți încerca, dacă doriți, dar, dacă o faceți, nu uitați să inserați caracterul punct și virgulă la sfârșit.)

În capitolele viitoare ale cărții, de regulă nu voi mai prezenta promptul `mysql>` sau caracterul punct și virgulă în instrucțiunile SQL. Motivul este acela că puteți emite interogări în contexte diferite de programul client `mysql` (de exemplu în scripturi Perl sau PHP), contexte în care nu apare nici un prompt și nu este necesar nici un caracter punct și virgulă. Voi indica situațiile în care se intenționează în mod specific ca dumneavoastră să introduceți o interogare în `mysql`.

Crearea bazei de date

Vom începe prin a crea baza de date `samp_db` și tabelele din interiorul acesteia, prin a popula tabelele sale și prin a efectua unele interogări simple asupra datelor incluse în aceste tabele.

Utilizarea unei baze de date implică mai multe etape:

1. Crearea (inițializarea) bazei de date
2. Crearea tabelelor din cadrul bazei de date
3. Interacțiunea cu tabelele prin inserția, regăsirea, modificarea sau ștergerea datelor.

Regăsirea datelor existente este, evident, cea mai frecventă operație efectuată cu o bază de date. Următoarele operații, în ordinea frecvenței, sunt inserția datelor noi și actualizarea sau ștergerea datelor existente. Mai puțin frecvente sunt operațiile de creare a tabelelor, iar operația cel mai puțin frecventă este crearea bazelor de date.

Noi pornim de la zero, deci trebuie să începem de la crearea bazei de date, operația cel mai puțin frecventă, și vom parcurge pas cu pas operațiile de creare a tabelelor și de inserție a datelor noastre inițiale, înainte de a ajunge să efectuăm operația cea mai comună: regăsirea datelor.

Pentru a crea o bază de date nouă, conectați-vă la server folosind `mysql` și apoi emiteți o instrucțiune `CREATE DATABASE` care specifică numele bazei de date:

```
mysql> CREATE DATABASE samp_db;
```

Va trebui să creați baza de date `samp_db` înainte de a putea crea oricare dintre tabelele care vor fi incluse în aceasta sau înainte de a efectua orice operații folosind conținutul acelor tabele.

Prin crearea bazei de date, devine aceasta baza de date curentă (prestabilită)? Nu, nu devine, așa cum se poate vedea prin executarea următoarei interogări:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
|            |
+------------+
```

Pentru ca baza de date `samp_db` să devină curentă, emiteți o instrucțiune `USE`:

```
mysql> USE samp_db
```

USE este una dintre puținele instrucțiuni care nu necesită un caracter punct și virgulă de terminare a instrucțiunii, deși îl puteți adăuga, dacă doriți. HELP este o altă instrucțiune de acest gen, iar dacă emiteți o instrucțiune HELP, aceasta vă va prezenta o listă cu toate instrucțiunile care nu necesită un caracter punct și virgulă.

După ce emiteți instrucțiunea USE, samp_db devine baza de date prestabilită:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| samp_db    |
+------------+
```

Cealaltă modalitate de a transforma o bază de date în bază de date curentă este de a o denumi în linia de comandă atunci când invocați mysql:

```
% mysql samp_db
```

Acesta este, de fapt, modul uzual de a denumi baza de date pe care doriți să o folosiți. Dacă aveți nevoie de parametri de conectare, specificați-i înainte de numele bazei de date. De exemplu, următoarele două comenzi îmi permit să mă conectez la baza de date samp_db de la gazda locală și de la sistemul pit-viper.snake.net:

```
% mysql -u paul -p samp_db
% mysql -h pit-viper.snake.net -u paul -p samp_db
```

Dacă nu se precizează altfel, toate exemplele care urmează vor presupune că, atunci când invocați mysql, specificați baza de date samp_db în linia de comandă pentru a o transforma în bază de date curentă. Dacă invocați mysql, dar uitați să specificați baza de date în linia de comandă, emiteți o instrucțiune USE samp_db.

Crearea tabelelor

În această secțiune, vom construi tabelele necesare pentru baza de date demonstrativă samp_db. Mai întâi, vom lua în considerare tabelele necesare pentru Liga istorică. Apoi, ne vom ocupa de tabelele necesare pentru proiectul de păstrare a evidenței rezultatelor școlare. Aceasta este partea unde unele cărți de baze de date încep să vorbească despre analiză și proiectare, diagrame entitate-relație, proceduri de normalizare și altele de acest gen. Există un loc pentru toate acestea, dar prefer să spun numai că trebuie să ne gândim puțin la modul în care va arăta baza noastră de date: care sunt tabelele pe care trebuie să le conțină, care trebuie să fie conținutul fiecărui tabel, precum și unele din aspectele implicate în decizia privind modul de reprezentare a datelor.

Opțiunile care se fac aici privind reprezentarea datelor nu sunt absolute. În alte situații, puteți opta la fel de bine să reprezentați date similare într-un mod diferit, în funcție de cerințele aplicațiilor dumneavoastră și de modurile în care doriți să vă utilizați datele.

Tabele pentru Liga istorică

Structura tabelelor pentru Liga istorică este relativ simplă:

- **Un tabel** `presedinte` (președinte). Acesta conține o înregistrare descriptivă pentru fiecare președinte al Statelor Unite. Vom avea nevoie de acesta pentru chestionarul electronic din situl Web al Ligii (analogul interactiv al chestionarului tipărit care apare în secțiunea pentru copii a buletinului de știri al Ligii).
- **Un tabel** `membru` (membru). Acesta este folosit pentru a păstra informații actuale despre fiecare membru al Ligii. Tabelul va fi folosit pentru crearea versiunilor tipărită și electronică a catalogului cu membri, pentru trimiterea de note automate de înștiințare pentru plata cotizațiilor și altele.

Tabelul presedinte

Tabelul presedinte este mai simplu, deci să discutăm despre el mai întâi. Acest tabel va conține unele informații biografice elementare despre fiecare președinte al Statelor Unite:

- **Nume.** Numele pot fi reprezentate într-un tabel în mai multe moduri. De exemplu, putem avea o singură coloană care conține întregul nume, respectiv coloane separate pentru nume și prenume. Este evident mai simplu să folosim o singură coloană, dar acest lucru vă dezavantajează în câteva moduri:
  - Dacă introduceți numele cu prenumele scris primul, nu le puteți sorta în funcție de numele de familie.
  - Dacă introduceți numele cu numele de familie scris primul, nu le puteți afișa cu prenumele scris primul.
  - Căutarea în funcție de nume este mai dificilă. De exemplu, pentru a căuta un anumit nume de familie, trebuie să folosiți un model și să căutați numele care corespund acelui model. Această operație este mai puțin eficientă și mai lentă decât căutarea unui anumit nume de familie.

Tabelul nostru presedinte va folosi coloane separate pentru nume și prenume, cu scopul de a evita aceste dezavantaje.

Coloana cu prenumele va reține, de asemenea, și al doilea prenume sau inițiala tatălui. Acest lucru nu va întrerupe nici o sortare pe care o vom executa, deoarece este puțin probabil să dorim sortarea în funcție de al doilea prenume (sau chiar după primul prenume). De asemenea, afișarea numelor trebuie să se producă în mod adecvat, indiferent dacă numele este scris în formatul "Bush, George W." sau "George W. Bush". Al doilea prenume urmează imediat după primul, în ambele situații.

Mai există o ușoară complicație. Un președinte (Jimmy Carter) are particula "Jr." la sfârșitul numelui său. Unde va fi introdusă aceasta? În funcție de formatul în care sunt afișate numele, numele acestui președinte va fi afișat sub forma "James E. Carter, Jr." sau "Carter, James E., Jr.". Particula "Jr." nu se asociază nici cu numele, nici cu prenumele, deci vom crea o altă coloană, care va conține sufixul numelui. Acest exemplu ilustrează modul în care chiar și o singură valoare poate cauza probleme atunci când încercați să determinați modul de reprezentare a datelor dumneavoastră. De asemenea, se arată de ce se recomandă să știți cât mai multe despre tipul valorilor datelor cu care veți lucra, înainte de a le insera într-o bază de date. Dacă aveți cunoștințe incomplete despre aspectul datelor dumneavoastră, s-ar putea să fiți obligat să modificați structura tabelului după ce ați început deja să-l folosiți. Acest fapt nu este în mod necesar un dezastru, dar în general este bine să-l evitați.

- **Locul nașterii (oraș și stat).** Ca și numele, și această informație poate fi reprezentată folosind una sau mai multe coloane. Este mai simplu de folosit o singură coloană, dar, ca în cazul numelui, coloanele separate vă permit efectuarea unor operații greu de executat în alte condiții. De exemplu, înregistrările cu președinții născuți într-un anumit stat sunt mai ușor de găsit dacă orașul și statul sunt enumerate separat.

- **Data naşterii şi a decesului.** Singura problemă specială în acest caz este aceea că nu putem cere să fie trecută data morţii, deoarece unii preşedinţi sunt încă în viaţă. MySQL furnizează o valoare `NULL` specială care înseamnă „fără valoare“, pe care o putem folosi în coloana cu data morţii, pentru a arăta că persoana respectivă trăieşte.

Tabelul `membru`

Tabelul `membru` pentru lista cu membrii Ligii istorice este similar cu tabelul `presedinte`, în sensul că fiecare înregistrare conţine informaţii descriptive elementare pentru o singură persoană. Fiecare înregistrare `membru` conţine însă mai multe coloane:

- **Nume.** Vom folosi aceeaşi reprezentare pe trei coloane ca în cazul tabelului `presedinte`: nume de familie, prenume (şi al doilea prenume, dacă există), sufix.
- **Număr de identificare.** Aceasta este o valoare unică, atribuită fiecărui membru atunci când acesta capătă calitatea respectivă. Liga nu a mai folosit niciodată numere de identificare, dar acum, când înregistrările sunt organizate mai sistematic, este o ocazie bună. (Eu anticipez că veţi găsi programul MySQL util şi că vă veţi gândi şi la alte moduri de a-l aplica înregistrărilor Ligii. Când acest lucru se va întâmpla, înregistrările din tabelul `membru` vor fi mai uşor de asociat cu alte tabele referitoare la membri pe care eventual le veţi crea, dacă folosiţi numere în locul numelor.)
- **Data expirării.** Membrii trebuie să-şi plătească periodic cotizaţia, pentru a evita pierderea calităţii de membru. Pentru unele aplicaţii, puteţi folosi data plăţii ultimei cotizaţii, dar acest lucru nu este convenabil pentru interesele Ligii. Cotizaţiile pot fi plătite pentru un număr variabil de ani (de regulă unul, doi, trei sau cinci ani), iar data plăţii ultimei cotizaţii nu indică momentul când trebuie plătită următoarea cotizaţie. De asemenea, Liga permite calitatea de membru pe viaţă. Putem reprezenta această informaţie printr-o dată din viitorul îndepărtat, dar valoarea `NULL` pare mai adecvată, deoarece „fără valoare“ poate fi asimilat logic cu „nu expiră niciodată“.
- **Adresă de e-mail.** Pentru membrii care au adresă de e-mail, aceasta le va permite să comunice cu uşurinţă unii cu alţii. Pe linia îndatoririlor dumneavoastră ca secretar al Ligii, aceasta vă va permite să trimiteţi înştiinţări de plată a cotizaţiei pe cale electronică şi nu prin poşta clasică, ceea ce este mai uşor decât să vă deplasaţi la poştă şi mai puţin costisitor. De asemenea, veţi putea folosi poşta electronică pentru a expedia membrilor conţinutul curent al intrărilor lor din catalog şi pentru a le cere să-şi actualizeze informaţiile, dacă este necesar.
- **Adresă poştală.** Aceasta este necesară pentru contactarea membrilor care nu dispun de poştă electronică (sau care nu răspund la mesajele primite). Vom folosi coloane pentru stradă, oraş, stat şi cod poştal. Coloana pentru stradă se poate dubla pentru a include numărul cutiei poştale, pentru acei membri care au o adresă precum P.O. Box 123 în loc de 123 Elm St.

Eu presupun că toţi membrii Ligii locuiesc în Statele Unite. Pentru instituţii ai căror membri au mai multe naţionalităţi, această presupunere este o simplificare excesivă, desigur. Dacă doriţi să utilizaţi adrese din mai multe ţări, veţi da peste unele aspecte mai ciudate legate de diversele formate de adrese folosite în diferite ţări. De exemplu, codul poştal nu este un standard internaţional, iar unele ţări nu au state, ci provincii.

- **Număr de telefon.** Ca şi câmpurile cu adrese, acesta este util pentru contactarea membrilor.
- **Cuvinte cheie speciale pentru zonele de interes.** Se presupune că fiecare membru este interesat de istoria Statelor Unite în general, dar probabil că membrii sunt interesaţi cu precădere de anumite domenii. Această coloană înregistrează zonele de interes respective, pe care membrii le pot folosi pentru a descoperi alţi membri cu interese similare.

Crearea tabelelor

Acum, suntem pregătiţi pentru crearea tabelelor aferente Ligii istorice. Pentru aceasta, vom folosi instrucţiunea `CREATE TABLE`, care are următoarea formă generală:

```
CREATE TABLE nume_tabel ( specificatii_coloane )
```

`nume_tabel` indică numele pe care doriţi să-l daţi tabelului. `specificatii_coloane` conţine specificaţiile pentru coloanele din tabel, inclusiv indexurile (dacă aveţi vreunul). Indexurile permit căutări mai rapide; vom discuta despre ele în capitolul 4, „Optimizarea interogărilor“.

Instrucţiunea `CREATE TABLE` pentru tabelul `presedinte` se prezintă astfel:

```
CREATE TABLE presedinte
(
    nume VARCHAR(15) NOT NULL,
    prenume VARCHAR(15) NOT NULL,
    sufix VARCHAR(5) NULL,
    oras VARCHAR(20) NOT NULL,
    stat VARCHAR(2) NOT NULL,
    data_nastere DATE NOT NULL,
    data_deces DATE NULL
)
```

Dacă doriţi să introduceţi personal instrucţiunea respectivă, invocaţi programul `mysql`, transformând `samp_db` în bază de date curentă:

```
% mysql samp_db
```

Apoi, introduceţi instrucţiunea `CREATE TABLE` aşa cum s-a arătat mai sus. (Nu uitaţi să adăugaţi caracterul punct şi virgulă la sfârşitul instrucţiunii; altfel, `mysql` nu va şti care este sfârşitul instrucţiunii.)

Pentru a crea tabelul `presedinte` folosind un fişier cu o descriere scrisă anterior din distribuţia bazei de date demonstrative (vezi „Obţinerea distribuţiei bazei de date demonstrative“), rulaţi această comandă din interpretor:

```
% mysql samp_db < create_president.sql
```

Indiferent de modul în care invocaţi `mysql`, specificaţi toţi parametrii de conexiune de care aveţi nevoie (numele gazdei, numele utilizatorului sau parola) în linia de comandă, înainte de numele bazei de date.

Fiecare specificaţie de coloană din instrucţiunea `CREATE TABLE` constă din numele coloanei, tipul (genul de valori pe care le va conţine coloana) şi eventual unele atribute ale coloanei.

Cele două tipuri de coloane folosite în tabelul `presedinte` sunt `VARCHAR` și `DATE`. `VARCHAR(`*n*`)` arată că acea coloană conține valori caracter (șir) de lungime variabilă, cu o lungime maximă de *n* caractere pentru fiecare valoare. Veți alege valoarea lui *n* în funcție de lungimea scontată a valorilor. Coloana `stat` este declarată ca `VARCHAR(2)`; nu avem nevoie de o valoare mai mare, dacă numele statelor sunt introduse folosind abrevieri din două caractere. Celelalte coloane cu valori șir trebuie să fie mai late, pentru a permite inserția unor valori mai lungi.

Celălalt tip de coloană pe care l-am folosit este `DATE`. Acest tip indică, deloc surprinzător, că acea coloană conține valori de tip dată calendaristică. Totuși, ceea ce vă poate surprinde este că datele sunt reprezentate începând cu anul. Formatul standard este `"AAAA-LL-ZZ"` (de exemplu `"1999-07-18"`). Acesta este standardul SQL ANSI pentru reprezentarea datelor calendaristice.

Singurele atribute de coloană pe care le folosim pentru tabelul `presedinte` sunt `NULL` (valorile pot lipsi), respectiv `NOT NULL` (valorile trebuie introduse). Majoritatea coloanelor au valoarea `NOT NULL`, deoarece avem întotdeauna o valoare pentru ele. Cele două coloane care pot avea valori `NULL` sunt `sufix` (majoritatea numelor nu au așa ceva) și `data_deces` (unii președinți sunt încă în viață, deci nu există o dată a decesului).

Instrucțiunea `CREATE TABLE` pentru tabelul `membru` se prezintă astfel:

```
CREATE TABLE membru
(
    nume VARCHAR(20) NOT NULL,
    prenume VARCHAR(20) NOT NULL,
    sufix VARCHAR (5) NULL,
    data_expirare DATE NULL DEFAULT "0000-00-00",
    email VARCHAR(100) NULL,
    strada VARCHAR(50) NULL,
    oras VARCHAR(50) NULL,
    stat VARCHAR(2) NULL,
    cod_postal VARCHAR(10) NULL,
    telefon VARCHAR(20) NULL,
    interese VARCHAR (255) NULL
)
```

Tastați această instrucțiune în `mysql` sau executați următoarea comandă de interpretor:

```
% mysql samp_db < create_member.sql
```

În ceea ce privește tipurile de coloane, tabelul `membru` nu este foarte interesant; fiecare coloană, cu o singură excepție, este creată sub forma unui șir de lungime variabilă. Excepția, `data_expirare`, este de tip `DATE`. Valoarea prestabilită a datei de expirare este `"0000-00-00"`, care este o valoare de tip non-`NULL`, ce indică faptul că nu a fost introdusă nici o dată corectă. Motivul este că `data_expirare` poate avea valoarea `NULL` pentru a arăta că un membru are această calitate pe o perioadă nelimitată de timp. Totuși, deoarece coloana poate avea valoarea `NULL`, aceasta devine valoarea prestabilită dacă nu specificați o altă valoare. Dacă ați creat un membru nou, dar ați uitat să specificați data expirării, membrul respectiv va deveni membru pe viață! Folosind valoarea prestabilită

`"0000-00-00"` vom evita această problemă. De asemenea, avem o valoare pe care o putem căuta periodic, pentru a găsi înregistrările unde data expirării nu a fost niciodată introdusă în mod adecvat.

Rețineți că am „omis" să inserez o coloană pentru numărul de identificare al unui membru. Aceasta deoarece am nevoie de un pretext ca să folosesc comanda `ALTER TABLE` mai târziu, iar omiterea acestei coloane furnizează un atare pretext.

Să verificăm că MySQL a creat tabelele conform așteptărilor. În `mysql`, emiteți următoarea interogare:

```
mysql> DESCRIBE presedinte;
```

Field	Type	Null	Key	Default	Extra
nume	varchar(15)				
prenume	varchar(15)				
sufix	varchar(5)	YES		NULL	
oras	varchar(20)				
stat	char(2)				
data_nastere	date			0000-00-00	
data_deces	date	YES		NULL	

Ca în cazul versiunii MySQL 3.23, datele de ieșire includ încă o coloană, care prezintă informații privind privilegiile de acces. Nu am prezentat aici coloana respectivă, deoarece rândurile devin prea lungi pentru a putea fi afișate cu ușurință.

Datele de ieșire sunt conform așteptărilor, cu excepția faptului că se precizează că tipul coloanei `stat` este `char(2)`. Ciudat; nu am declarat coloana ca fiind de tip `VARCHAR(2)`? Ba da, am declarat-o, dar MySQL a modificat „pe șest" tipul din `VARCHAR` în `CHAR`. Motivul acestei acțiuni este legat de eficiența spațiului de stocare pentru coloanele scurte cu date de tip caracter, despre care nu vom discuta în detaliu aici. Dacă doriți detalii, consultați paragrafele dedicate instrucțiunii `ALTER TABLE` din capitolul 3, „Sintaxa și utilizarea SQL în MySQL", care conțin detalii. Pentru noi, în acest moment, nu există nici o diferență între cele două tipuri.

Dacă emiteți o interogare `DESCRIBE membru`, atunci `mysql` vă va prezenta informații similare pentru tabelul `membru`.

Interogarea `DESCRIBE` este utilă atunci când uitați numele unei coloane din tabel, când trebuie să știți tipul coloanei, când trebuie să știți care este lățimea coloanei și altele. De asemenea, este util pentru a afla ordinea în care MySQL stochează coloanele în rândurile tabelului. Ordinea respectivă este importantă atunci când folosiți instrucțiunile `INSERT` sau `LOAD DATA`, care presupun afișarea valorilor din coloane în ordinea prestabilită a coloanelor.

`DESCRIBE` se poate abrevia sub forma `DESC` sau, dacă preferați să tastați mai mult, `SHOW COLUMNS FROM nume_tabel` este sinonim cu `DESCRIBE nume_tabel`.

Și dacă uitați numele tabelelor dumneavoastră? Atunci puteți folosi `SHOW TABLES`. Pentru baza de date `samp_db`, care conține cele două tabele pe care le-am creat până acum, datele de ieșire ale comenzii se prezintă astfel:

```
mysql> SHOW TABLES;
+-------------------+
| Tables_in_samp_db |
+-------------------+
| membru            |
| presedinte        |
+-------------------+
```

Dacă nu reușiți nici măcar să vă reamintiți numele bazei de date, invocați `mysql` fără a specifica un nume al bazei de date în linia de comandă; apoi emiteți o interogare `SHOW DATABASES`:

```
mysql> SHOW DATABASES;
+-----------+
| Database  |
+-----------+
| menajerie |
| mysql     |
| samp_db   |
| test      |
+-----------+
```

Lista cu baze de date variază de la un server la altul, dar trebuie ca în listă să apară cel puțin `samp_db` și `mysql`; ultima conține tabelele de acordare, care controlează privilegiile de acces MySQL.

Interogările `DESCRIBE` și `SHOW` au echivalente în linie de comandă pe care le puteți folosi din interpretor:

`% mysqlshow`	Afișează toate bazele de date, ca `SHOW DATABASES`
`% mysqlshow nume_db`	Afișează tabelele din baza de date dată, ca `SHOW TABLES`
`% mysqlshow nume_db nume_tabel`	Afișează coloanele din tabelul dat, ca `DESCRIBE nume_tabel`

Tabele pentru proiectul de evidență a rezultatelor școlare

Pentru a vedea care sunt tabelele de care avem nevoie pentru proiectul de evidență a rezultatelor școlare, să examinăm modul în care se pot scrie punctajele atunci când folosiți un catalog pe hârtie. Figura 1.2 prezintă o pagină din catalogul dumneavoastră. Conținutul principal al acestei pagini este o matrice pentru înregistrarea punctajelor. De asemenea, se mai află și alte informații necesare pentru înțelegerea punctajelor. Numele elevilor și numerele de identificare sunt prezentate în partea de jos a matricei. (Pentru simplitate, au fost prezentate numai notele a patru elevi.) De-a lungul părții de sus a matricei, scrieți datele chestionarelor și ale testelor. Figura arată că ați dat chestionare pe 3, 6, 16 și 23 septembrie, respectiv teste pe 9 septembrie, respectiv 1 octombrie.

Pentru a păstra evidența acestor tipuri de informații folosind o bază de date, avem nevoie de un tabel `puncte`. Ce trebuie să conțină înregistrările din acest tabel? Simplu. Pentru fiecare rând, avem nevoie de numele elevului, data chestionarului sau a testului și punctajul obținut. Figura 1.3 prezintă aspectul unor note din catalog atunci când sunt reprezentate într-un tabel ca acesta. (Datele sunt scrise în modul în care le reprezintă MySQL, respectiv `"AAAA-LL-ZZ"`.)

Elevi		Puncte						
		C	C	T	C	C	T	
ID	Nume	9/3	9/6	9/9	9/16	9/23	10/1	...
1	Billy	14	10	73	14	15	67	...
2	Missy	17	10	68	17	14	73	...
3	Johnny	15	10	78	12	17	82	...
4	Jenny	14	13	85	13	19	79	...
...	...	...	...	...	...	...	...	...

Figura 1.2 Exemplu de catalog.

Tabelul `puncte`

Nume	Data	Puncte
Billy	1999-09-23	15
Missy	1999-09-23	14
Johnny	1999-09-23	17
Jenny	1999-09-23	19
Billy	1999-10-01	67
Missy	1999-10-01	73
Johnny	1999-10-01	82
Jenny	1999-10-01	79

Figura 1.3 Aspectul inițial al tabelului `puncte`.

Totuși, la configurarea tabelului în acest mod apare o problemă. Se pare că tabelul omite unele informații. De exemplu, dacă examinăm înregistrările din figura 1.3, nu ne putem da seama dacă punctajele au fost obținute la un chestionar sau la un test. Este important să știm tipurile de punctaje la determinarea notelor finale, dacă chestionarele și testele au ponderi diferite. Putem încerca să deducem tipul examinării din plaja de rezultate de la o anumită dată (de regulă, la chestionare se obțin mai puține puncte decât la teste), dar nu este convenabil, deoarece ne bazăm pe deducție și nu pe date explicite.

Se poate face deosebire între punctaje menționând tipul acestora în fiecare înregistrare, de exemplu prin adăugarea unei coloane în tabelul `puncte` care conține literele `T` sau `C` în fiecare rând, pentru a indica „test" sau „chestionar", așa cum se poate vedea în figura 1.4. Acest procedeu are avantajul că tipul punctajului devine explicit în cadrul datelor. Dezavantajul este că aceste informații sunt oarecum redundante. Remarcați că, pentru toate înregistrările cu o anumită dată, coloana cu tipul punctajului are întotdeauna aceeași valoare. Punctajele pentru 23 septembrie au toate tipul `C`, iar rezultatele pentru 1 octombrie sunt toate de tipul `T`. Neplăcut. Dacă înregistrăm un set de punctaje pentru un chestionar sau pentru un test în acest mod, nu numai că vom scrie aceeași dată pentru fiecare înregistrare nouă din set, dar vom scrie mereu și mereu același tip de punctaj. Uf... Cine să scrie toate aceste informații redundante?

Să încercăm o reprezentare alternativă. În loc de a înregistra tipurile de punctaje în tabelul `puncte`, le vom deduce din datele calendaristice. Putem păstra o listă cu date calendaristice și o putem folosi pentru a păstra evidența tipului de „examinare" (chestionar sau test) care a avut loc la fiecare dată. Apoi, putem determina dacă un anumit punctaj a fost obținut la un chestionar sau la un test prin combinarea acestuia cu informațiile din lista noastră cu evenimente: se stabilește o echivalență între data din tabelul `puncte` cu

data din tabelul `eveniment` pentru a determina tipul de punctaj. Figura 1.5 prezintă această machetă de tabel și demonstrează modul în care această asociere funcționează pentru o înregistrare din tabelul `puncte` care poartă data de 23 septembrie. Prin coroborarea înregistrării cu înregistrarea echivalentă din tabelul `eveniment`, vedem că punctajul a fost obținut la un chestionar.

Tabelul `puncte`

Nume	Data	Puncte	Tip
Billy	1999-09-23	15	C
Missy	1999-09-23	14	C
Johnny	1999-09-23	17	C
Jenny	1999-09-23	19	C
Billy	1999-10-01	67	T
Missy	1999-10-01	73	T
Johnny	1999-10-01	82	T
Jenny	1999-10-01	79	T

Figura 1.4 Macheta tabelului `puncte`, revizuită pentru a include tipurile de punctaje.

Tabelul `puncte`

Nume	Data	Puncte
Billy	1999-09-23	15
Missy	1999-09-23	14
Johnny	1999-09-23	17
Jenny	1999-09-23	19
Billy	1999-10-01	67
Missy	1999-10-01	73
Johnny	1999-10-01	82
Jenny	1999-10-01	79

Tabelul `eveniment`

Data	Tip
1999-09-03	C
1999-09-06	C
1999-09-09	T
1999-09-16	C
1999-09-23	C
1999-10-01	T

Figura 1.5 Tabelele `puncte` și `eveniment`, legate prin dată.

Este mult mai bine decât să încercați a deduce tipul punctajului „pe ghicite"; în schimb, obținem tipul direct din datele înregistrate în mod explicit în baza noastră de date. De asemenea, este o opțiune de preferat față de înregistrarea tipurilor de punctaje în tabelul `puncte`, deoarece trebuie să înregistrăm fiecare tip o singură dată.

Totuși, dacă sunteți ca mine, când auziți pentru prima dată de acest lucru (combinarea informațiilor din mai multe tabele), vă gândiți: „Mda, bună idee, dar e multă muncă să faci mereu toate căutările alea; nu complicăm lucrurile?"

Într-un fel, aveți dreptate. Folosirea a două liste cu înregistrări *este* mai complicată decât folosirea unei singure liste. Dar să mai privim o dată la catalog (vezi figura 1.2). Nu aveți *deja* două seturi de înregistrări? Țineți cont de următoarele aspecte:

- Păstrați evidența punctajelor folosind celulele din matricea tabelului `puncte`, unde fiecare celulă este indexată de numele elevului și dată (în jos pe lateral și de-a lungul părții superioare a matricei). Acesta reprezintă un set de înregistrări; este analog cu conținutul tabelului `punctaj`.
- Cum procedați pentru a ști tipul de eveniment pe care îl reprezintă fiecare dată? Ați scris un `T` sau un `C` mic deasupra fiecărei date! Astfel, păstrați și evidența asocierii între dată și tipul punctajului, de-a lungul părții de sus a matricei. Acesta reprezintă un al doilea set de înregistrări; este analog cu conținutul tabelului `eveniment`.

Cu alte cuvinte, chiar dacă nu conștientizați acest lucru, în realitate utilizați catalogul într-un mod absolut similar cu cel propus de mine, de păstrare a informațiilor în două tabele. Singura diferență reală este că acele două categorii de informații nu sunt separate în mod atât de explicit în catalog.

Pagina din catalog ilustrează într-un fel modul în care noi gândim informațiile, precum și dificultatea determinării modului de a plasa informațiile într-o bază de date: noi tindem să integrăm diferitele categorii de informații și să le interpretăm ca pe un întreg. Bazele de date procedează altfel, motiv pentru care par uneori artificiale și nenaturale. Tendința noastră naturală de a unifica informațiile îngreunează uneori până și sesizarea faptului că avem mai multe tipuri de date, nu doar unul singur. Din această cauză, încercarea de a „gândi ca o bază de date" în ceea ce privește modul de reprezentare a datelor poate fi o mare problemă.

O cerință impusă tabelului `eveniment` de macheta prezentată în figura 1.5 este aceea ca datele să fie unice, deoarece data este folosită pentru a corela înregistrările din tabelele `puncte` și `eveniment`. Cu alte cuvinte, nu puteți da două chestionare în aceeași zi, respectiv un chestionar și un test. Dacă o faceți, veți avea două seturi de înregistrări în tabelul `puncte` și două înregistrări în tabelul `eveniment`, toate cu aceeași dată, și nu veți mai putea stabili o corespondență între înregistrările din cele două tabele.

Aceasta este o problemă care nu va apărea niciodată dacă într-o zi survine o singură examinare, dar putem presupune în mod corect că acest lucru nu se va întâmpla? În aparență răspunsul este afirmativ; la urma urmelor, nu sunteți atât de sadic încât să dați un chestionar și un test în aceeași zi. Dar sper că mă veți ierta dacă sunt sceptic. Am auzit deseori oameni care spun despre datele lor: „Acest caz ciudat nu se va întâmpla niciodată." Apoi, se dovedește că acel caz ciudat *chiar* se produce cu o anumită ocazie și de regulă este necesar să vă re-proiectați tabelele pentru a remedia problema pe care a cauzat-o acel caz ciudat.

Este mai bine să anticipați posibilele probleme și modul de a le rezolva. Deci, să presupunem că uneori sunteți în situația de a înregistra două seturi de punctaje pentru aceeași zi. Cum ne putem descurca? Așa cum se va vedea, nu este o problemă chiar atât de dificilă. Cu o mică modificare în modul de dispunere a datelor, apariția mai multor evenimente la o anumită dată nu va provoca necazuri:

1. Adăugați o coloană la tabelul `eveniment` și folosiți-o pentru a-i atribui un număr unic fiecărei înregistrări din tabel. Astfel, fiecare eveniment va avea propriul său număr de identificare, deci coloana va avea numele `eveniment_id`. (Dacă pare ciudat, catalogul din figura 1.2 are deja această proprietate: identificatorul de eveniment este similar numărului coloanei din matricea cu punctaje din catalog. Se poate ca numărul să nu fie scris în mod explicit și denumit „identificator de eveniment", dar este chiar un identificator de evenimente.)
2. Când inserați punctaje în tabelul `puncte`, înregistrați identificatorul de eveniment în locul datei.

Rezultatul acestor modificări este prezentat în figura 1.6. Acum, coroborați tabelele `puncte` și `eveniment` folosind identificatorul de eveniment în locul datei și utilizați tabelul `eveniment` pentru a determina nu numai tipul fiecărui punctaj, dar și data la care a fost obținut acesta. De asemenea, elementul unic din tabelul `eveniment` nu mai trebuie

să fie data, ci identificatorul de eveniment. Aceasta înseamnă că puteți avea zece teste și chestionare în aceeași zi; puteți păstra evidența lor corectă în scriptele dumneavoastră. (Indiscutabil, elevii dumneavoastră vor fi extrem de încântați să afle acest lucru.)

Din păcate, din punct de vedere al operatorului uman, macheta de tabel din figura 1.6 pare mai puțin satisfăcătoare decât cele anterioare. Tabelul `puncte` este mai abstract, deoarece conține mai puține coloane cu o semnificație evidentă. Macheta de tabel prezentată anterior în figura 1.4 era ușor de examinat și de înțeles, deoarece tabelul `puncte` avea coloane atât pentru date, cât și pentru tipurile de punctaj. Tabelul `puncte` actual, prezentat în figura 1.6, nu are coloane pentru nici una din aceste informații, ceea ce pare serios deplasat față de normal. Cine vrea să citească un tabel `puncte` care conține „identificatoare de eveniment"? Pentru noi, această informație nu are o semnificație prea importantă.

Tabelul `puncte`

Nume	eveniment_id	Puncte
Billy	5	15
Missy	5	14
Johnny	5	17
Jenny	5	19
Billy	6	67
Missy	6	73
Johnny	6	82
Jenny	6	79

Tabelul `eveniment`

eveniment_id	Data	Tip
1	1999-09-03	C
2	1999-09-06	C
3	1999-09-09	T
4	1999-09-16	C
5	1999-09-23	C
6	1999-10-01	T

Figura 1.6 Tabelele `puncte` și `eveniment`, legate după identificatorul de eveniment.

Acum ați ajuns la o răscruce. Sunteți interesat de posibilitatea de a putea păstra evidența electronică a rezultatelor școlare și de a nu trebui să efectuați tot felul de calcule manuale obositoare atunci când atribuiți notele. Dar, după ce vă gândiți la modul în care veți reprezenta informațiile despre punctaje într-o bază de date, sunteți descurajat de aspectul abstract și discontinuu al informației reprezentate în acel mod.

Natural, aceasta duce la o întrebare: „N-ar fi bine să nu folosesc deloc o bază de date? Poate că MySQL nu este pentru mine." Așa cum puteți intui, voi răspunde negativ la această întrebare, deoarece altfel cartea de față ar ajunge rapid la final. Dar, atunci când vă gândiți cum să faceți un lucru anume, este bine să aveți în vedere diferite alternative și să vă întrebați dacă n-ar fi mai bine să folosiți un sistem de baze de date, cum este SQL, sau altceva, cum ar fi un program de calcul tabelar:

- Catalogul are rânduri și coloane, ca și o foaie de calcul tabelar. De aceea, o foaie de calcul tabelar și un catalog sunt foarte asemănătoare atât vizual, cât și conceptual.
- Într-o foaie de calcul tabelar se pot efectua calcule, deci puteți însuma punctajele fiecărui elev folosind un câmp de calcul. Este puțin mai greu să acordați o pondere diferită chestionarelor și testelor, dar nu imposibil.

Pe de altă parte, dacă doriți să examinați numai o parte a datelor dumneavoastră (numai chestionarele sau numai testele, de exemplu), să faceți comparații (între notele băieților și cele ale fetelor) sau să afișați informații de sumar într-un mod flexibil, datele problemei se schimbă. O foaie de calcul tabelar nu funcționează atât de bine, în timp ce sistemele de baze de date relaționale execută cu ușurință aceste operații.

Un alt element care trebuie avut în vedere este aceea că natura abstractă și discontinuă a datelor reprezentate într-o bază de date relațională nu este, la urma urmelor, o problemă atât de mare. Trebuie să vă gândiți la acea reprezentare atunci când configurați baza de date, astfel încât să nu vă dispuneți datele într-un mod lipsit de sens în raport cu felul în care doriți să le utilizați. Totuși, după ce ați determinat modul de reprezentare, veți conta pe motorul de baze de date pentru adunarea și reprezentarea datelor dumneavoastră într-un mod semnificativ. Nu veți mai privi datele ca pe o adunătură de componente fără legătură între ele.

De exemplu, când regăsiți punctaje din tabelul `puncte`, nu doriți să vedeți identificatoare de eveniment; doriți să vedeți date. Aceasta nu este o problemă. Baza de date va căuta datele din tabelul `eveniment` în funcție de identificatorul de eveniment și vi le va prezenta. De asemenea, puteți dori să vedeți dacă punctajele au fost obținute la chestionare sau teste. Nici aceasta nu este o problemă. Baza de date va căuta tipurile de punctaje în același mod: folosind identificatorul de eveniment. Nu uitați, acesta este „punctul forte" al unui sistem de baze de date relaționale ca MySQL: corelarea unui lucru cu altul pentru a extrage informații din mai multe surse și de a vă prezenta datele pe care doriți să le vedeți. În cazul datelor noastre de evidență a rezultatelor școlare, MySQL este cel care adună informațiile la un loc folosind identificatorii de eveniment, pentru ca dumneavoastră să nu mai efectuați această operație.

Acum, pentru a da o mică idee „în avans" cu privire la modul în care cereți sistemului MySQL să execute această corelație între lucruri, să presupunem că doriți să vedeți punctajele obținute la data de 23 septembrie 1999. Interogarea pentru extragerea punctajelor aferente unui eveniment produs la o anumită dată se prezintă astfel:

```
SELECT puncte.nume, eveniment.data, puncte.puncte, eveniment.tip
FROM puncte, eveniment
WHERE eveniment.data = "1999-09-23"
AND puncte.eveniment_id = eveniment.eveniment_id
```

Cam de speriat, este? Această interogare regăsește numele elevului, data, punctajul și tipul punctajului prin alăturarea (corelarea) înregistrărilor din tabelul `puncte` cu înregistrările din tabelul `eveniment`. Rezultatul arată astfel:

Nume	Data	Puncte	Tip
Billy	1999-09-2315	15	C
Missy	1999-09-2314	14	C
Johnny	1999-09-2317	17	C
Jenny	1999-09-2319	19	C

Formatul acestor informații vă „sună" cunoscut? Ar fi și cazul; este același cu macheta tabelului prezentat în figura 1.4! Nu trebuie să cunoașteți identificatorul de eveniment pentru a obține acest rezultat. Specificați data care vă interesează și lăsați sistemul MySQL să determine înregistrările din tabelul `puncte` care corespund datelor respective. Dacă v-ați întrebat în ce măsură din cauza aspectului abstract și discontinuu se „pierde" ceva în ceea ce privește preluarea informațiilor dintr-o bază de date într-o formă semnificativă pentru noi, răspunsul este: nu se pierde nimic.

Desigur, după ce ați examinat această interogare, vă mai puteți pune și altă întrebare. Instrucțiunea pare cam lungă și complicată; nu pare o cantitate cam mare de muncă numai pentru regăsirea punctajelor obținute la o anumită dată? Ba da, este. Totuși, există metode de a evita tastarea mai multor rânduri de instrucțiuni SQL de fiecare dată când doriți să emiteți o interogare. În general, determinați o singură dată modul de a executa o interogare ca aceea și apoi o stocați, pentru a o putea repeta cu ușurință atunci când este necesar. Vom vedea cum se procedează în secțiunea „Sugestii pentru interacțiunea cu MySQL".

În realitate, am cam exagerat punând pe tapet acea interogare. Și totuși interogarea respectivă este, credeți sau nu, ceva mai simplă decât cea pe care o vom folosi efectiv pentru extragerea punctajelor. Motivul este că trebuie să mai facem o modificare în structura tabelului. În loc de a înregistra numele elevului în tabelul `puncte`, vom folosi un identificator de elev unic. (Adică vom folosi valoarea din coloana ID a catalogului în locul valorii din coloana Nume.) Apoi vom crea un alt tabel, denumit `elev`, care conține coloanele `nume` și `elev_id` (Figura 1.7).

Tabelul `elev`

Nume	Sex	elev_id
Billy	M	1
Missy	F	2
Johnny	M	3
Jenny	F	4

Tabelul `puncte`

elev_id	eveniment_id	Puncte
1	5	15
2	5	14
3	5	17
4	5	19
5	6	67
6	6	73
7	6	82
8	6	79

Tabelul `eveniment`

eveniment_id	Data	Tip
1	1999-09-03	C
2	1999-09-06	C
3	1999-09-09	T
4	1999-09-16	C
5	1999-09-23	C
6	1999-10-01	T

Figura 1.7 Tabelele `elev`, `puncte` și `eveniment`, legate prin identificatorul de elev și identificatorul de eveniment.

De ce să facem această modificare? În primul rând pentru că pot fi doi elevi cu același nume. Utilizarea unui număr unic de identificare a elevului vă ajută să faceți diferența dintre punctajele elevilor. (Este exact același lucru ca și diferențierea între punctajele obținute la un test și la un chestionar date în aceeași zi prin utilizarea în locul datei a unui identificator unic de eveniment.)

După această modificare a machetei tabelului, interogarea pe care o vom folosi efectiv pentru extragerea punctajelor aferente unei anumite date devine puțin mai complexă:

```
SELECT elev.nume, eveniment.data, puncte.puncte, eveniment.tip
FROM eveniment, puncte, elev
WHERE eveniment.data = "1999-09-23"
AND eveniment.eveniment_id = puncte.eveniment_id
AND puncte.elev_id = elev.elev_id
```

Dacă vă îngrijorează faptul că nu puteți determina imediat semnificația acelei interogări, liniștiți-vă. Majoritatea oamenilor nu sunt în stare de așa ceva. Vom revedea interogarea după ce vom continua parcurgerea acestui manual, dar diferența între acum și mai târziu este că mai târziu o veți înțelege. Nu, nu mă țin de bancuri.

Din figura 1.7, veți observa că am adăugat ceva la tabelul `elev` care nu exista în catalog: o coloană pentru sex. Aceasta va permite efectuarea unor operații simple, precum numărarea băieților și a fetelor din clasă, respectiv mai complexe, precum compararea punctajelor băieților cu cele ale fetelor.

Suntem aproape gata cu tabelele pentru proiectul de evidență a rezultatelor școlare. Mai avem nevoie de un singur tabel, pentru înregistrarea absențelor în vederea determinării prezenței la curs. Conținutul acestuia este relativ simplu: un identificator numeric al elevului și o dată (vezi figura 1.8). Fiecare rând din tabel arată că elevul respectiv a fost absent la acea dată. La sfârșitul perioadei de notare, vom apela la caracteristicile de numărare ale sistemului MySQL pentru a rezuma conținutul tabelului, astfel încât să știm de câte ori a fost absent fiecare elev.

Tabelul `absente`

elev_id	Data
2	1999-09-02
4	1999-09-15
2	1999-09-20

Figura 1.8 Tabelul `absente`.

Suntem gata de a crea tabelele noastre de evidență a rezultatelor școlare, acum când știm cum vor arăta acestea.

Instrucțiunea `CREATE TABLE` pentru tabelul `elev` arată astfel:

```
CREATE TABLE elev
(
    nume VARCHAR(20) NOT NULL,
    sex ENUM('F','M') NOT NULL,
    elev_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
)
```

Tastați această instrucțiune în `mysql` sau executați următoarea comandă de interpretor:

```
% mysql samp_db < create_student.sql
```

Instrucțiunea `CREATE TABLE` creează un tabel denumit `elev` cu trei coloane: `nume`, `sex` și `elev_id`.

`nume` este o coloană șir de lungime variabilă, care poate conține maximum 20 de caractere. Această reprezentare a numelor este mai simplă decât cea folosită pentru tabelele Ligii istorice, folosind o singură coloană în locul unor coloane separate pentru nume și prenume. Aceasta deoarece știu dinainte că nici un exemplu de interogare privind evidența rezultatelor școlare nu va trebui să efectueze vreo operație care s-ar preta mai bine la două coloane. (Da, trișez. Recunosc.)

`sex` determină dacă un elev este băiat sau fată. Este o coloană de tip `ENUM` (enumerare), ceea ce înseamnă că poate lua numai una din valorile enumerate explicit în specificația coloanei, în speță `M` sau `F`, adică de sex masculin sau feminin. `ENUM` este util când aveți un set limitat de valori pe care le poate conține o coloană. Am fi putut folosi în schimb `CHAR(1)`, dar `ENUM` specifică mai exact care pot fi valorile din coloană. Dacă emiteți o instrucțiune `DESCRIBE nume_tabel` pentru un tabel care conține o coloană `ENUM`, MySQL vă va indica exact care sunt valorile posibile.

Apropo, valorile dintr-o coloană ENUM nu trebuie să fie neapărat compuse dintr-un singur caracter. Coloana tip ar fi putut fi declarată astfel: ENUM ('feminin','masculin').

elev_id este o coloană de tip întreg care va conține numere unice de identificare a elevilor. În mod normal, veți primi numerele de identificare a elevilor de la o sursă centrală, cum ar fi secretariatul școlii, dar noi vom folosi valori inventate. Declarația coloanei elev_id are numeroase părți, chiar dacă nu conține decât numere:

- INT arată că în coloană sunt incluse numere întregi (valori fără parte fracționară)
- UNSIGNED interzice utilizarea numerelor negative.
- NOT NULL înseamnă că valoarea coloanei trebuie completată. (Nici un elev nu poate fi lipsit de un număr de identificare.)
- AUTO_INCREMENT este un atribut special în MySQL și funcționează astfel: dacă valoarea din coloana elev_id lipsește (sau este NULL), atunci când creați o nouă înregistrare în tabelul elev, MySQL generează automat un număr unic cu o unitate mai mare decât valoarea maximă care se află în coloană la momentul respectiv. Vom folosi acest fapt la încărcarea tabelului elev, specificând valori numai pentru coloanele nume și sex și permițând sistemului MySQL să genereze automat valoarea coloanei elev_id.
- PRIMARY KEY arată că acea coloană este indexată pentru căutări rapide și că fiecare valoare din coloană trebuie să fie unică. Aceasta ne împiedică să folosim din greșeală același identificator de două ori, ceea ce este o proprietate necesară a numerelor de identificare a elevilor. (Nu numai atât; MySQL impune ca fiecare coloană AUTO_INCREMENT să aibă un indice unic.)

Dacă nu înțelegeți toate chestiile astea despre AUTO_INCREMENT și PRIMARY_KEY, gândiți-vă că ele reprezintă o modalitate magică de a genera numere de identificare pentru fiecare elev. Nu contează valorile în sine, atâta vreme cât sunt unice.

Notă: Dacă aveați de gând să luați identificatoarele numerice ale elevilor de la secretariat (fără să le generați automat), puteți declara coloana elev_id în același mod, cu deosebirea că veți omite atributul AUTO_INCREMENT.

Tabelul eveniment se prezintă astfel:

```
CREATE TABLE eveniment
(
    data DATE NOT NULL
    tip ENUM('T','C') NOT NULL,
    eveniment_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
)
```

Tastați instrucțiunea de mai sus în mysql sau executați următoarea comandă de interpretor:

```
% mysql samp_db < create_event.sql
```

Toate coloanele sunt declarate ca NOT NULL, deoarece nu poate lipsi valoarea nici uneia dintre ele.

Coloana data conține o valoare DATE standard MySQL, în format "AAAA-LL-ZZ" (cu anul scris primul).

tip reprezintă tipul punctajului. Ca și sex din tabelul elev, tip este o coloană de tip enumerare. Valorile permise sunt T și C, reprezentând „test", respectiv „chestionar".

eveniment_id este o coloană AUTO_INCREMENT, similară coloanei elev_id din tabelul elev. Utilizarea opțiunii AUTO_INCREMENT ne permite să generăm cu ușurință valori unice ale identificatorilor de eveniment. Ca în cazul coloanei elev_id din tabelul elev, valorile particulare sunt mai puțin importante decât unicitatea lor.

Tabelul puncte arată astfel:

```
CREATE TABEL puncte
(
    elev_id INT UNSIGNED NOT NULL,
    eveniment_id INT UNSIGNED NOT NULL,
    punctaj INT NOT NULL,
    PRIMARY KEY (eveniment_id, elev_id)
)
```

Tastați acea instrucțiune în mysql sau executați următoarea comandă de interpretor:

```
% mysql samp_db < create_score.sql
```

puncte este o coloană INT (întreg). Cu alte cuvinte, se presupune că valorile punctajelor sunt întotdeauna întregi. Dacă doriți să permiteți includerea unor punctaje precum 58,5, care conțin o parte fracționară, veți folosi unul din tipurile de coloană cu virgulă mobilă, precum FLOAT sau DECIMAL.

Coloanele elev_id și eveniment_id sunt întregi care reprezintă elevul și evenimentul pentru care a fost acordat fiecare punctaj. Folosindu-le pentru a stabili legături între tabelele elev și eveniment, vom putea determina numele elevului și data evenimentului. De asemenea, am transformat combinația dintre cele două coloane într-o cheie primară (PRIMARY KEY). Astfel, ne asigurăm că nu vom avea punctaje repetate pentru un elev la un test sau chestionar dat. De asemenea, modificarea ulterioară a unui punctaj este mai simplă. De exemplu, când un punctaj a fost introdus incorect, putem elimina înregistrarea veche atunci când inserăm înregistrările noi, folosind instrucțiunea MySQL REPLACE. Nu este necesar să folosim o instrucțiune DELETE combinată cu INSERT; MySQL execută automat această operație.

Rețineți unicitatea combinației între coloanele eveniment_id și elev_id. În tabelul puncte, nici o valoare nu este intrinsec unică. Vor exista mai multe înregistrări de punctaje pentru fiecare valoare eveniment_id (câte una pentru fiecare elev), respectiv mai multe înregistrări pentru fiecare valoare elev_id (câte una pentru fiecare chestionar și test).

Tabelul absente pentru determinarea prezenței la curs se prezintă astfel:

```
CREATE TABLE absente
(
    elev_id INT UNSIGNED NOT NULL,
    data DATE NOT NULL,
    PRIMARY KEY (elev_id, data)
)
```

Tastați instrucțiunea respectivă în mysql sau executați următoarea comandă de interpretor:

```
% mysql samp_db < create_absence.sql
```

Coloanele `elev_id` și `data` sunt ambele declarate `NOT NULL` pentru a interzice absența valorilor. Vom transforma combinația dintre cele două coloane într-o cheie primară, astfel încât să nu creăm din greșeală înregistrări dublate. La urma urmelor, nu-i frumos să pui absență unui elev de două ori într-o zi!

Adăugarea de noi înregistrări

În acest moment, baza noastră de date și tabelele au fost create, iar în secțiunea următoare, „Regăsirea informațiilor", veți afla cum se pot extrage datele din baza de date. Mai întâi însă, trebuie să inserăm unele înregistrări în tabele.

Există numeroase modalități de a adăuga date într-o bază de date. Puteți insera manual înregistrări într-un tabel emițând instrucțiuni `INSERT`. De asemenea, puteți adăuga înregistrări citindu-le dintr-un fișier, fie sub formă de valori de date brute pe care le încărcați folosind instrucțiunea `LOAD DATA` sau utilitarul `mysqlimport`, fie în formă de instrucțiuni `INSERT` redactate anterior, pe care le introduceți în `mysql`.

Această secțiune ilustrează fiecare metodă de inserție a înregistrărilor în tabelele dumneavoastră. Ce aveți dumneavoastră de făcut este să vă jucați cu diferitele metode, pentru a vedea cum funcționează. Apoi, deplasați-vă la sfârșitul secțiunii și rulați comenzile pe care le găsiți acolo pentru a șterge conținutul tabelelor și pentru a le reîncărca. Astfel, vă veți asigura că tabelele conțin aceleași înregistrări la care am lucrat când am scris secțiunea următoare, iar dumneavoastră veți obține aceleași rezultate.

Să începem să adăugăm rezultate folosind `INSERT`, o instrucțiune SQL pentru care specificați tabelul în care doriți să inserați un rând de date și valorile care vor fi plasate în rândul respectiv.

Instrucțiunea `INSERT` are mai multe forme:

- Puteți specifica valori pentru toate coloanele:

  `INSERT INTO` *`nume_tabel`* `VALUES(`*`valoare1,valoare2,...`*`)`

  De exemplu:

  ```
  mysql> INSERT INTO elev VALUES('Mihai','M',NULL);
  mysql> INSERT INTO eveniment VALUES("1999-9-3","C",NULL);
  ```

  Cuvântul `INTO` este opțional începând de la MySQL 3.22.5. (Acest lucru este valabil și pentru celelalte forme ale instrucțiunii `INSERT`.) Lista `VALUES` trebuie să conțină o valoare pentru fiecare coloană din tabel, în ordinea în care sunt stocate coloanele în tabel. (În mod normal, aceasta este ordinea în care au fost specificate coloanele la crearea tabelului. Folosiți `DESCRIBE nume_tabel` pentru a afla ordinea, dacă nu o cunoașteți cu exactitate.)

  Puteți încadra între ghilimele valorile de tip șir și dată în MySQL, folosind ghilimele simple sau duble. Valorile `NULL` din exemplul precedent sunt destinate coloanelor `AUTO_INCREMENT` din tabelele `elev` și `eveniment`. (Inserția unei „valori lipsă" determină generarea următorului număr `elev_id` sau `eveniment_id`.)

  Versiunile MySQL începând de la 3.22.5 vă permit să inserați mai multe rânduri într-un tabel cu o singură instrucțiune `INSERT`, prin specificarea mai multor liste de valori:

  `INSERT INTO` *`nume_tabel`* `VALUES(...),(...),...`

  De exemplu:

  ```
  mysql> INSERT INTO elev VALUES('Ana','F',NULL),("Mihai","M",NULL);
  ```

  Acest procedeu necesită introducerea de la tastatură a unei cantități mai reduse de text decât cea impusă de mai multe instrucțiuni `INSERT`, iar execuția sa de către server este mai eficientă.

- Puteți denumi coloanele cărora doriți să le atribuiți valori, apoi enumerați valorile. Acest procedeu este util când doriți să creați o înregistrare pentru care numai câteva coloane trebuie configurate inițial.

  `INSERT INTO` *`nume_tabel (nume_col1,nume_col2,...)`* `VALUES(`*`valoare1,valoare2,...`*`)`

  De exemplu:

  ```
  mysql> INSERT INTO membru (nume,prenume) VALUES('Tache','Stefan');
  ```

  De la MySQL 3.22.5, această formă a instrucțiunii `INSERT` permite de asemenea mai multe liste cu valori:

  ```
  mysql> INSERT INTO elev (nume,sex) VALUES('Ana','F'),("Mihai","M");
  ```

  Coloanele care nu sunt specificate în lista de coloane primesc o valoare prestabilită.

- De la MySQL 3.22.10, puteți denumi coloanele și valorile sub forma `nume_coloana=valoare`.

  `INSERT INTO` *`nume_tabel`* `SET` *`nume_col1=valoare1, nume_col2=valoare2,...`*

  De exemplu:

  ```
  mysql> INSERT INTO membru SET nume='Tache',prenume='Stefan';
  ```

  Toate coloanele care nu sunt specificate în clauza `SET` primesc o valoare prestabilită.

  Nu puteți insera mai multe rânduri folosind această formă a instrucțiunii `INSERT`.

O altă metodă pentru încărcarea înregistrărilor într-un tabel este citirea valorilor datelor direct dintr-un fișier. Puteți încărca înregistrări în acest mod cu ajutorul instrucțiunii `LOAD DATA` sau cu utilitarul `mysqlimport`.

Instrucțiunea `LOAD DATA` se comportă ca un încărcător de mare capacitate, care citește date dintr-un fișier. Folosiți această instrucțiune din interiorul programului `mysql`:

```
mysql> LOAD DATA LOCAL INFILE "membru.txt" INTO TABLE membru;
```

Această instrucțiune citește conținutul fișierului de date `membru.txt` localizat în catalogul dumneavoastră curent din gazda programului client și o trimite serverului, pentru a fi încărcată în tabelul `membru`.

`LOAD DATA LOCAL` nu va avea efect dacă versiunea dumneavoastră de MySQL este anterioară versiunii 3.22.15, deoarece posibilitatea de citire a fișierelor de la client a fost adăugată la instrucțiunea `LOAD DATA` începând de la versiunea respectivă. (Fără cuvântul cheie `LOCAL`, fișierul trebuie localizat pe gazda serverului și dumneavoastră aveți nevoie de un privilegiu de acces la server, de care majoritatea utilizatorilor MySQL nu dispun.)

În mod prestabilit, instrucțiunea `LOAD DATA` presupune că valorile coloanelor sunt separate prin tabulatori și că rândurile se încheie cu un caracter linie nouă. De asemenea, presupune că valorile există în ordinea în care coloanele sunt stocate în tabel. Este posibilă citirea fișierelor în alte formate sau specificarea unei alte ordini a coloanelor. Vezi rubrica aferentă comenzii `LOAD DATA` din Anexa D, „Referință de sintaxă SQL", pentru mai multe detalii.

Utilitarul `mysqlimport` este o interfață în linie de comandă pentru instrucțiunea `LOAD DATA`. Acest utilitar se invocă din interpretor și generează automat o instrucțiune `LOAD DATA`:

```
% mysqlimport --local samp_db membru.txt
```

`mysqlimport` generează automat o instrucțiune `LOAD DATA`, care determină încărcarea fișierului `membru.txt` în tabelul `membru`. Acest lucru nu este posibil dacă programul dumneavoastră MySQL este anterior versiunii 3.22.15, deoarece opțiunea `--local` necesită `LOAD DATA LOCAL`. Ca și în cazul programului `mysql`, dacă trebuie să specificați parametri de conexiune, indicați-i în linia de comandă, anterior numelui bazei de date.

`mysqlimport` extrage numele tabelului din numele fișierului de date. (Ca nume al tabelului, se folosesc toate caracterele, până la primul punct din numele fișierului.) De exemplu, `membru.txt` va fi încărcat în tabelul `membru`, iar `presedinte.txt` va fi încărcat în tabelul `presedinte`. Dacă aveți mai multe fișiere de încărcat într-un singur tabel, alegeți numele fișierelor cu atenție; în caz contrar, `mysqlimport` nu va folosi numele corect de tabel. Cu nume ca `membru1.txt` sau `membru2.txt`, `mysqlimport` va presupune că numele tabelelor sunt `membru1`, respectiv `membru2`. Totuși, puteți folosi nume precum `membru.1.txt` și `membru.2.txt`, respectiv `membru.txt1` și `membru.txt2`.

După ce ați testat aceste metode de adăugare a înregistrărilor, trebuie să ștergeți conținutul tabelelor și apoi să le reîncărcați, astfel încât conținutul lor să fie cel presupus în secțiunea următoare.

Din interpretorul dumneavoastră, executați următoarele comenzi:

```
% mysql samp_db < insert_president.sql
% mysql samp_db < insert_member.sql
% mysql samp_db < insert_student.sql
% mysql samp_db < insert_score.sql
% mysql samp_db < insert_event.sql
% mysql samp_db < insert_absence.sql
```

Fiecare fișier conține o instrucțiune `DELETE` pentru ștergerea tuturor înregistrărilor pe care le-ați inserat în tabel, urmată de un set de comenzi `INSERT` pentru reinițializarea conținutului tabelului. Dacă nu doriți să scrieți fiecare din aceste comenzi în parte, încercați următoarea comandă:

```
% cat insert_*.sql | mysql samp_db
```

Regăsirea informațiilor

Acum, tabelele noastre au fost create și încărcate cu date, deci haideți să vedem ce putem face cu acele date. Instrucțiunea `SELECT` vă permite să regăsiți și să afișați informații din tabelele dumneavoastră, într-o manieră oricât de generală sau de specifică doriți. Puteți afișa întregul conținut al unui tabel:

```
SELECT * FROM presedinte
```

Sau puteți selecta o singură coloană a unui singur rând:

```
SELECT data_nastere FROM presedinte WHERE nume = "Eisenhower"
```


Instrucțiunea `SELECT` are numeroase clauze (părți) pe care le combinați conform necesităților pentru a regăsi informația care vă interesează. Fiecare dintre aceste clauze poate fi simplă sau complexă, deci instrucțiunile `SELECT` în totalitatea lor pot fi simple sau complexe. Totuși, puteți sta liniștit; nu veți găsi în această carte interogări lungi de o pagină, pentru care aveți nevoie de o oră pentru a le pricepe. (Când găsesc într-un material interogări lungi de-un cot, în general le omit fără mustrări de conștiință și bănuiesc că și dumneavoastră procedați la fel.)

Forma generală a instrucțiunii `SELECT` este:

```
SELECT ce se selecteaza
FROM tabel sau tabele
WHERE conditii pe care trebuie sa le satisfaca datele
```

Nu uitați că SQL este un limbaj cu format liber; deci, când vă scrieți propriile dumneavoastră interogări `SELECT`, nu este necesar să inserați salturile la linie nouă acolo unde le-am inserat eu.

Pentru a scrie o instrucțiune `SELECT`, specificați ce anume doriți să regăsiți și apoi unele clauze opționale. Clauzele prezentate mai sus (`FROM`, `WHERE`) sunt cel mai frecvent folosite, deși mai pot fi specificate și altele, cum sunt `GROUP BY`, `ORDER BY` și `LIMIT`.

Clauza `FROM` este de regulă prezentă, dar acest lucru nu este necesar dacă nu selectați date din tabele. De exemplu, interogarea următoare afișează pur și simplu valorile unor expresii care pot fi calculate fără referire la vreun tabel, deci nu este necesară nici o clauză `FROM`:

```
mysql> SELECT 2+2, "Salut, lume", VERSION();
```

2+2	Salut, lume	VERSION()
4	Salut, lume	3.23.0-alpha-log

Când folosiți o clauză `FROM` pentru a specifica un tabel de unde vor fi regăsite datele, forma cea mai „generică" a instrucțiunii `SELECT` este aceea prin care se regăsesc toate datele. Pentru aceasta, folosiți caracterul *, care este o abreviere de la „toate coloanele". Interogarea următoare regăsește toate rândurile din tabelul `elev` și le afișează:

```
mysql> SELECT * FROM elev;
```

nume	sex	elev_id
Megan	F	1
Joseph	M	2
Kyle	M	3
Katie	F	4
...		

Coloanele sunt returnate în ordinea în care MySQL le-a stocat în tabel. Aceasta este ordinea în care coloanele sunt afișate atunci când se emite o instrucțiune `DESCRIBE elev`. Punctele de suspensie de la sfârșitul exemplului arată că interogarea returnează mai multe rânduri decât cele prezentate.

Puteți denumi în mod explicit coloana sau coloanele pe care doriți să le vedeți. Pentru a selecta numai numele elevilor, procedați astfel:

```
mysql> SELECT nume FROM elev;
```

nume
Megan
Joseph
Kyle
Katie
...

Dacă precizați mai mult de o coloană, separați numele coloanelor prin virgule. Instrucțiunea următoare este echivalentă cu `SELECT * FROM elev`, dar fiecare coloană este specificată în mod explicit:

```
mysql> SELECT nume,sex,elev_id FROM elev;
```

nume	sex	elev_id
Megan	F	1
Joseph	M	2
Kyle	M	3
Katie	F	4
...		

Puteți denumi coloanele în orice ordine:

```
SELECT nume,elev_id FROM elev
SELECT elev_id,nume FROM elev
```

Puteți chiar specifica o coloană de mai multe ori, dacă doriți, deși acest lucru este, în general, inutil.

Numele coloanelor nu sesizează diferența între majuscule și minuscule în MySQL. Următoarele interogări sunt echivalente:

```
SELECT nume,elev_id FROM elev
SELECT NUME,ELEV_ID FROM elev
SELECT nUmE, eLeV_iD FROM elev
```

Numele bazelor de date și ale tabelelor pot sesiza diferența între majuscule și minuscule; depinde de sistemul de fișiere folosit pe gazda serverului. Un server care rulează în UNIX tratează numele tabelelor și ale bazelor de date ca sensibile la diferența între majuscule și minuscule, deoarece numele de fișiere UNIX sunt sensibile la această diferență. Numele de fișiere Windows nu sesizează această diferență, deci un server care rulează în Windows nu va trata numele tabelelor și ale bazelor de date ca sensibile la diferența dintre majuscule și minuscule.

MySQL vă permite să selectați coloane din mai multe tabele simultan. Vom discuta despre aceasta în secțiunea „Regăsirea informațiilor din mai multe tabele".

Specificarea criteriilor de regăsire

Pentru a limita setul de înregistrări regăsite de instrucțiunea `SELECT`, folosiți o clauză `WHERE` care specifică criteriile pentru selectarea rândurilor. Puteți selecta rânduri examinând acele valori din coloane care satisfac diverse criterii.

Puteți căuta valori numerice:

```
mysql> SELECT * FROM puncte WHERE puncte > 95;
```

elev_id	eveniment_id	puncte
5	3	97
18	3	96
1	6	100
5	6	97
11	6	98
16	6	98

Alternativ, puteți căuta valori șir. (Rețineți: comparațiile între șiruri nu sunt, în mod normal, sensibile la diferența între majuscule și minuscule.)

```
mysql> SELECT nume, prenume FROM presedinte
    -> WHERE nume="ROOSEVELT";
```

prenume	nume
Roosevelt	Theodore
Roosevelt	Franklin D.

```
mysql> SELECT nume, prenume FROM presedinte
    -> WHERE nume="roosevelt";
```

prenume	nume
Roosevelt	Theodore
Roosevelt	Franklin D.

Sau puteți căuta valori ale unor date calendaristice:

```
mysql> SELECT nume, prenume, data_nastere FROM presedinte
    -> WHERE data_nastere < "1750-1-1";
```

nume	prenume	data_nastere
Washington	George	1732-02-22
Adams	John	1735-10-30
Jefferson	Thomas	1743-04-13

Sau puteți căuta o combinație de valori:

```
mysql> SELECT nume, prenume, data_nastere, stat FROM presedinte
    -> WHERE data_nastere < "1750-1-1" AND (stat="VA" OR stat="MA");
```

nume	prenume	data_nastere	stat
Washington	George	1732-02-22	VA
Adams	John	1735-10-30	MA
Jefferson	Thomas	1743-04-13	VA

Expresiile din clauzele `WHERE` pot folosi operatori aritmetici, ca în tabelul 1.1, operatori de comparație, ca în tabelul 1.2, respectiv operatori logici, ca în tabelul 1.3. De asemenea, puteți folosi paranteze pentru a grupa părți ale unei expresii. Operațiile pot fi efectuate folosind constante, coloane de tabel și apeluri la funcții. Vom avea ocazia de a folosi numeroase funcții MySQL în interogări pe parcursul acestui manual, dar ele sunt

mult prea numeroase pentru a le prezenta aici. Vezi Anexa C, „Referință de operatori și funcții“, pentru o listă completă.

Tabelul 1.1 Operatori aritmetici

Operator	Semnificație
+	Adunare
-	Scădere
*	Înmulțire
/	Împărțire

Tabelul 1.2 Operatori de comparație

Operator	Semnificație
<	Mai mic
<=	Mai mic sau egal
=	Egal
!= sau <>	Diferit
>=	Mai mare sau egal
>	Mai mare

Tabelul 1.3 Operatori logici

Operator	Semnificație
AND	ȘI logic
OR	SAU logic
NOT	Negație logică

Când formulați o interogare care necesită operatori logici, fiți atent să nu confundați semnificația operatorului logic AND cu modul în care folosim cuvântul „și“ în vorbirea zilnică. Să presupunem că doriți să găsiți „președinții născuți în statul Virginia și președinții născuți în statul Massachusetts“. Observați modul de utilizare a lui „și“; înseamnă aceasta că veți scrie interogarea după cum urmează?:

```
SELECT nume, prenume, stat FROM presedinte
WHERE stat="VA" AND stat="MA"
```

Nu, deoarece această interogare înseamnă „selectează președinții născuți atât în Virginia, cât și în Massachusetts“, ceea este lipsit de sens. În limbajul uzual puteți exprima interogarea folosind cuvântul „și“, dar în SQL cele două condiții se leagă folosind OR (sau):

```
mysql> SELECT nume, prenume, stat FROM presedinte
    -> WHERE stat="VA" OR stat="MA";
```


nume	prenume	stat
Washington	George	VA
Adams	John	MA
Jefferson	Thomas	VA
Madison	James	VA
Monroe	James	VA
Adams	John Quincy	MA
Harrison	William H.	VA
Tyler	John	VA
Taylor	Zachary	VA
Wilson	Woodrow	VA
Kennedy	John F	MA
Bush	George W.	MA

Acesta este un aspect la care trebuie să fiți atent, nu numai când vă formulați propriile dumneavoastră interogări, dar și când scrieți interogări pentru alte persoane. Cel mai bine este să ascultați cu atenție ce anume doresc să afle acestea, dar nu este necesar să le transcrieți descrierile în SQL folosind aceiași operatori logici. Pentru exemplul descris anterior, echivalentul în limba vorbită al interogării este „Selectează președinții care s-au născut în Virginia sau Massachusetts.“

Valoarea NULL

Valoarea NULL este specială; deoarece înseamnă „fără valoare“, nu o puteți compara cu valori cunoscute în modul în care comparați una cu alta două valori cunoscute. Dacă încercați să folosiți NULL cu operatorii uzuali de comparație aritmetică, rezultatul este nedefinit:

```
mysql> SELECT NULL < 0, NULL = 0, NULL != 0, NULL > 0;
```

NULL < 0	NULL = 0	NULL != 0	NULL > 0
NULL	NULL	NULL	NULL

De fapt, nu puteți compara NULL nici măcar cu sine însuși, deoarece rezultatul comparației între două valori necunoscute nu poate fi cunoscut:

```
mysql> SELECT NULL = NULL, NULL!= NULL;
```

NULL = NULL	NULL != NULL
NULL	NULL

Pentru a efectua căutări ale valorilor NULL, trebuie să folosiți o sintaxă specială. În loc de a folosi = sau != pentru a testa egalitatea sau inegalitatea, folosiți IS NULL sau IS NOT NULL.

De exemplu, deoarece am reprezentat data decesului pentru președinții aflați în viață sub forma NULL, le putem găsi după cum urmează:

```
mysql> SELECT nume, prenume, FROM presedinte
    -> WHERE data_deces IS NULL;
```

nume	prenume
Ford	Gerald R
Carter	James E.
Reagan	Ronald W.
Bush	George W.
Clinton	William J.

Pentru a găsi numele care au un sufix, folosiți `IS NOT NULL`:

```
mysql> SELECT nume, prenume, sufix FROM presedinte
    -> WHERE sufix IS NOT NULL;
```

nume	prenume	sufix
Carter	James E.	Jr.

MySQL 3.23 și versiunile superioare au un operator de comparație special, specific programului MySQL, și anume <=>, care este adevărat chiar și pentru comparațiile între valori `NULL`. Cele două interogări precedente pot fi re-scrise folosind acest operator după cum urmează:

```
mysql> SELECT nume, prenume, FROM presedinte
    -> WHERE data_deces <=> NULL;
```

nume	prenume
Ford	Gerald R
Carter	James E.
Reagan	Ronald W.
Bush	George W.
Clinton	William J.

```
mysql> SELECT nume, prenume, sufix FROM presedinte
    -> WHERE NOT (sufix <=> NULL);
```

nume	prenume	sufix
Carter	James E.	Jr.

Sortarea rezultatelor unei interogări

Uneori se observă că, dacă se emite o interogare `SELECT * FROM nume_tabel` asupra unui tabel după încărcarea în tabel a unui set inițial de date, rândurile sunt regăsite în aceeași ordine în care au fost inserate. Acest fapt nu este întotdeauna adevărat. Dacă ștergeți și inserați rânduri după încărcarea inițială a tabelului, se observă modificarea ordinii în care serverul returnează rândurile tabelului. (Prin ștergerea înregistrărilor, în tabel sunt inserate „goluri" de spațiu nefolosit, goluri pe care MySQL încearcă să le umple ulterior, când inserați înregistrări noi.)

În mod prestabilit, când inserați rânduri, serverul nu garantează ordinea în care vor fi returnate rândurile. Pentru a sorta rândurile, folosiți o clauză `ORDER BY`:

```
mysql> SELECT nume, prenume FROM presedinte
    -> ORDER BY nume;
```


nume	prenume
Adams	John
Adams	John Quincy
Arthur	Chester A.
Buchanan	James
...	

Puteți specifica dacă sortarea unei coloane se va face în ordine ascendentă sau descendentă folosind cuvintele cheie `ASC` sau `DESC` după numele coloanelor din clauza `ORDER BY`. De exemplu, pentru a sorta numele președinților în ordine inversă (descrescătoare), folosiți `DESC` astfel:

```
mysql> SELECT nume, prenume FROM presedinte
    -> ORDER BY nume DESC;
```

nume	prenume
Wilson	Woodrow
Washington	George
Van Buren	Martin
Tyler	John
...	

Ordinea ascendentă este cea prestabilită dacă nu specificați `ASC` sau `DESC` pentru un nume de coloană într-o clauză `ORDER BY`.

Dacă sortați o coloană care poate conține valori `NULL`, toate valorile `NULL` apar la începutul coloanei, respectiv la sfârșit, dacă sortați în ordine descendentă.

Rezultatele interogării pot fi sortate pe mai multe coloane, iar fiecare coloană poate fi sortată în ordine ascendentă sau descendentă, independent de celelalte coloane. Interogarea următoare regăsește rânduri din tabelul `presedinte`, le sortează în sens descendent în funcție de statul de naștere și după numele de familie în cadrul fiecărui stat:

```
mysql> SELECT nume, prenume, stat FROM presedinte
    -> ORDER BY stat DESC, nume ASC;
```

nume	prenume	stat
Arthur	Chester A.	VT
Coolidge	Calvin	VT
Harrison	William H.	VA
Jefferson	Thomas	VA
Madison	James	VA
Monroe	James	VA
Taylor	Zachary	VA
Tyler	John	VA
Washington	George	VA
Wilson	Woodrow	VA
Eisenhower	Dwight D.	TX
Johnson	Lyndon B.	TX
...		

Limitarea rezultatelor unei interogări

Când o interogare returnează mai multe rânduri, dar dumneavoastră doriți să vedeți numai câteva, clauza `LIMIT` este utilă, mai ales în conjuncție cu `ORDER BY`. MySQL vă permite să limitați datele de ieșire ale unei interogări la primele *n* rânduri ale rezultatului returnat. Interogarea următoare selectează pe primii cinci născuți președinți americani:

```
mysql> SELECT nume, prenume, data_nastere FROM presedinte
    -> ORDER BY data_nastere LIMIT 5;
```

nume	prenume	data_nastere
Washington	George	1732-02-22
Adams	John	1735-10-30
Jefferson	Thomas	1743-04-13
Madison	James	1751-03-16
Monroe	James	1758-04-28

Dacă sortați în ordine inversă, folosind `ORDER BY data_nastere DESC`, veți obține pe ultimii cinci născuți președinți americani.

De asemenea, `LIMIT` vă permite să extrageți o secțiune de înregistrări din mijlocul unui set de rezultate. Pentru aceasta, trebuie să specificați două valori. Prima valoare este numărul înregistrării inițiale din setul de rezultate pe care doriți să-l vedeți. (Prima înregistrare este numerotată cu 0, nu 1.) Cea de-a doua valoare indică numărul de înregistrări care vor fi returnate. Interogarea următoare este similară cu precedenta, dar returnează cinci înregistrări, începând de la al unsprezecelea rând al rezultatului:

```
mysql> SELECT nume, prenume, data_nastere FROM presedinte
    -> ORDER BY data_nastere LIMIT 10,5;
```

nume	prenume	data_nastere
Tyler	John	1790-03-29
Buchanan	James	1791-04-23
Polk	James K.	1795-11-02
Fillmore	Millard	1800-01-07
Pierce	Franklin	1804-11-23

Începând cu MySQL versiunea 3.23.2, puteți ordona rezultatele interogărilor în funcție de o formulă. De exemplu, pentru a extrage o înregistrare selectată aleator din tabelul `presedinte`, folosiți `ORDER BY RAND()` în conjuncție cu `LIMIT`:

```
mysql> SELECT nume, prenume FROM presedinte
    -> ORDER BY RAND() LIMIT 1;
```

nume	prenume
McKinley	William

Calculul și denumirea valorilor din coloanele de ieșire

Majoritatea interogărilor precedente au produs date de ieșire prin regăsirea valorilor din tabele. MySQL vă mai permite să calculați o valoare din coloana de ieșire ca rezultat al unei expresii. Expresiile pot fi simple sau complexe. Interogarea următoare evaluează o expresie simplă (o constantă) și o expresie mai complexă, care implică numeroase operații aritmetice și câteva apeluri la funcții:

```
mysql> SELECT 17, FORMAT(SQRT(3*3+4*4),0);
```

17	FORMAT(SQRT(3*3+4*4),0)
17	5

De asemenea, expresiile se pot referi la coloanele din tabele:

```
mysql> SELECT CONCAT(prenume," ",nume),CONCAT(oras,", ",stat)
    -> FROM presedinte;
```

CONCAT(prenume," ",nume)	CONCAT(oras,", ",stat)
George Washington	Wakefield, VA
John Adams	Braintree, MA
Thomas Jefferson	Albemarle County, VA
James Madison	Port Conway, VA
...	

Această interogare formatează numele președinților sub forma unui singur șir, prin concatenarea numelui și a prenumelui separate printr-un spațiu și formatează locul nașterii sub forma orașelor și statelor separate prin virgulă.

Când folosiți o expresie pentru a calcula valoarea unei coloane, expresia este folosită pentru antetul coloanei. Aceasta poate duce la o coloană foarte lată, dacă expresia este lungă (așa cum o arată interogarea precedentă). Pentru a rezolva problema, capului de coloană i se poate atribui un nume, folosind construcția `AS nume`. Asemenea nume se numesc aliasuri de coloană. Rezultatul interogării anterioare poate deveni astfel mai semnificativ:

```
mysql> SELECT CONCAT(prenume," ",nume), AS Nume,
    -> CONCAT(oras,", ",stat) As Loculnasterii
    -> FROM presedinte;
```

Nume	Loculnasterii
George Washington	Wakefield, VA
John Adams	Braintree, MA
Thomas Jefferson	Albemarle County, VA
James Madison	Port Conway, VA
...	

În cazul în care aliasul coloanei conține spații, acestea vor trebui inserate între ghilimele:

```
mysql> SELECT CONCAT(prenume," ",nume), AS "Numele presedintelui"
    -> CONCAT(oras,", ",stat) As "Locul nasterii"
    -> FROM presedinte;
```

Numele presedintelui	Locul nasterii
George Washington	Wakefield, VA
John Adams	Braintree, MA
Thomas Jefferson	Albemarle County, VA
James Madison	Port Conway, VA
...	

Lucrul cu date calendaristice

Principalul aspect pe care trebuie să-l reţineţi atunci când lucraţi cu date în MySQL este că acestea sunt întotdeauna reprezentate începând cu anul. Data de 27 iulie 1999 este reprezentată sub forma "1999-07-27". Datele *nu* se scriu nici sub forma "07-27-1999" şi nici sub forma "27-07-1999", aşa cum poate că eraţi obişnuit să le scrieţi.

MySQL oferă numeroase moduri de efectuare a operaţiilor cu date. Unele dintre acestea sunt următoarele:

- Sortare după dată. (Deja am văzut această operaţie de mai multe ori.)
- Căutarea unei anumite date sau a unui domeniu de date.
- Extragerea unei părţi dintr-o valoare de tip dată, precum anul, luna sau ziua.
- Calcularea diferenţei între date.
- Calcularea unei date prin adunarea sau scăderea unui interval dintr-o altă dată.

În continuare sunt prezentate exemple cu aceste operaţii.

Pentru a căuta anumite date, fie în funcţie de valoarea exactă, fie prin comparaţie cu o altă valoare, comparaţi o coloană DATA cu valoarea care vă interesează:

```
mysql> SELECT * FROM eveniment WHERE data="1999-10-01";
```

data	tip	eveniment_id
1999-10-01	T	6

```
mysql> SELECT nume, prenume, data_deces
    -> FROM presedinte
    -> WHERE data_deces >= "1970-01-01" AND data_deces < "1980-01-01";
```

nume	prenume	data_deces
Truman	Harry S.	1972-12-26
Johnson	Lyndon B.	1973-01-22

Pentru a testa sau pentru a regăsi părţi ale datelor calendaristice, puteţi folosi funcţii precum YEAR() (an), MONTH() (lună) sau DAYOFMONTH() (ziua din lună). De exemplu, pot găsi preşedinţii care s-au născut în aceeaşi lună ca mine (martie) căutând datele a căror valoare a lunii este egală cu 3:

```
mysql> SELECT nume, prenume, data_nastere
    -> FROM presedinte WHERE MONTH(data_nastere) = 3;
```


nume	prenume	data_nastere
Madison	James	1751-03-16
Jackson	Andrew	1767-03-15
Tyler	John	1790-03-29
Cleveland	Grover	1837-03-18

De asemenea, interogarea poate fi scrisă şi în funcţie de numele lunii:

```
mysql> SELECT nume, prenume, data_nastere
    -> FROM presedinte
    -> WHERE MONTHNAME(data_nastere) = "March";
```

nume	prenume	data_nastere
Madison	James	1751-03-16
Jackson	Andrew	1767-03-15
Tyler	John	1790-03-29
Cleveland	Grover	1837-03-18

Pentru a fi mai exact – specificând chiar ziua naşterii – pot combina interogările pentru MONTH() şi DAYOFMONTH() pentru a găsi preşedinţii născuţi în ziua mea de naştere:

```
mysql> SELECT nume, prenume, data_nastere
    -> FROM presedinte WHERE MONTH(data_nastere) = 3
    -> AND DAYOFMONTH(data_nastere) = 29;
```

nume	prenume	data_nastere
Tyler	John	1790-03-29

Acesta este tipul de interogare pe care îl folosiţi pentru a genera liste cu persoanele născute într-o anumită zi, listă pe care o vedeţi în secţiunea de divertisment a ziarelor. Totuşi, nu trebuie să scrieţi o anumită zi, aşa cum s-a procedat în interogarea anterioară. Pentru a căuta preşedinţii născuţi astăzi, indiferent de ziua din an în care ne găsim, comparaţi data naşterii lor cu valoarea CURRENT_DATE (dată curentă):

```
SELECT nume, prenume, data_nastere
FROM presedinte WHERE MONTH(data_nastere) = MONTH(CURRENT_DATE)
AND DAYOFMONTH(data_nastere) = DAYOFMONTH(CURRENT_DATE)
```

Puteţi scădea o dată dintr-o altă dată. Aceasta vă permite să găsiţi intervalul dintre date, care este util pentru determinarea vârstelor. De exemplu, pentru a determina preşedintele care a trăit cel mai mult, se scade data naşterii din data decesului. Pentru aceasta, convertiţi data_nastere şi data_deces în zile folosind funcţia TO_DAYS(), luaţi diferenţa şi o împărţiţi la 365 pentru a obţine vârsta aproximativă, în ani:

```
mysql> SELECT nume, prenume, data_nastere, data_deces,
    -> FLOOR((TO_DAYS(data_deces) - TO_DAYS(data_nastere))/365
    -> AS varsta FROM presedinte WHERE data_deces IS NOT NULL
    -> ORDER BY varsta DESC LIMIT 5;
```

nume	prenume	data_nastere	data_deces	varsta
Adams	John	1735-10-30	1826-07-04	90
Hoover	Herbert C.	1874-08-10	1964-10-20	90
Truman	Harry S.	1884-05-08	1972-12-26	88
Madison	James	1751-03-16	1836-06-28	85
Jefferson	Thomas	1743-04-13	1826-07-04	83

Funcția `FLOOR()` folosită în această interogare elimină toate părțile fracționare din valoarea vârstei, pentru a genera un întreg.

Stabilirea diferenței între date este de asemenea utilă pentru determinarea distanței în raport cu o anumită dată de referință. Astfel, vă puteți da seama care sunt membrii Ligii istorice care trebuie să-și plătească în curând cotizația. Calculați diferența între datele de expirare și data curentă, iar dacă aceasta este mai mică decât o anumită valoare de prag, înseamnă că plata cotizației va fi în curând necesară. Următoarea interogare îi găsește pe membrii care trebuie să-și achite cotizația în termen de 60 de zile:

```
SELECT nume, prenume, data_expirare FROM membru
WHERE (TO_DAYS(data_expirare) - TO_DAYS(CURRENT_DATE)) < 60
```

Începând de la MySQL 3.22, puteți folosi `DATE_ADD()` sau `DATE_SUB()` pentru a calcula o dată pornind de la alta. Aceste funcții preiau o dată și un interval și produc o dată nouă. De exemplu:

```
mysql> SELECT DATE_ADD("1970-1-1", INTERVAL 10 YEAR);
```

DATE_ADD("1970-1-1", INTERVAL 10 YEAR)
1980-01-01

```
mysql> SELECT DATE_SUB("1970-1-1", INTERVAL 10 YEAR);
```

DATE_SUB("1970-1-1", INTERVAL 10 YEAR)
1960-01-01

O interogare prezentată anterior în această secțiune selecta președinții care au decedat în timpul anilor '70, folosind date literale pentru extremitățile domeniului de selecție. Acea interogare poate fi reformulată folosind o dată de pornire literală și o dată finală calculată pe baza datei de pornire și a unui interval:

```
mysql> SELECT nume, prenume, data_deces
    -> FROM presedinte
    -> WHERE data_deces >= "1970-1-1"
    -> AND data_deces < DATE_ADD("1970-1-1", INTERVAL 10 YEAR)
```

nume	prenume	data_deces
Truman	Harry S.	1972-12-26
Johnson	Lyndon B.	1973-01-22

Interogarea pentru plata cotizației membrilor poate fi scrisă folosind `DATE_ADD()`:

```
SELECT nume, prenume, data_expirare FROM membru
WHERE data_expirare < DATE_ADD(CURRENT_DATE, INTERVAL 60 DAY)
```


Anterior în acest capitol, a fost prezentată o interogare pentru determinarea pacienților unui medic stomatolog care nu s-au mai prezentat la control în ultimul timp:

```
SELECT nume, prenume, ultima_vizita FROM pacient
WHERE ultima_vizita < DATE_SUB(CURRENT_DATE, INTERVAL 6 MONTH)
```

În acel moment, interogarea respectivă nu era prea semnificativă. Acum este?

Stabilirea corespondenței cu un model

MySQL vă permite să căutați valori care corespund unui anumit model. Astfel, puteți selecta înregistrări fără a furniza o valoare exactă. Pentru a executa o operație de stabilire a corespondenței cu un model, folosiți operatori speciali (`LIKE` și `NOT LIKE`) și specificați un șir care conține caractere de înlocuire. Caracterul `_` corespunde oricărui caracter, iar `%` corespunde oricărei secvențe de caractere (inclusiv o secvență vidă). Stabilirea corespondenței cu un model folosind `LIKE` și `NOT LIKE` nu face diferența între majuscule și minuscule.

Acest model găsește numele care încep cu W (sau cu w, dacă ar exista):

```
mysql> SELECT nume, prenume FROM presedinte
    -> WHERE nume LIKE "W%";
```

nume	prenume
Washington	George
Wilson	Woodrow

Această corespondență cu un model este eronată:

```
mysql> SELECT nume, prenume FROM presedinte
    -> WHERE nume = "W%";
Empty set (0.00 sec)
```

Interogarea ilustrează o eroare comună, și anume folosirea unui model cu un operator aritmetic de comparație. Unica posibilitate ca această operație să reușească este ca în coloană să se afle chiar șirul `"W%"` sau `"w%"`.

Acest model corespunde numelor de familie care conțin litera `W` sau `w` oriunde în interiorul numelui:

```
mysql> SELECT nume, prenume FROM presedinte
    -> WHERE nume LIKE "%W%";
```

nume	prenume
Washington	George
Wilson	Woodrow
Eisenhower	Dwight D.

Acest model corespunde numelor de familie care conțin exact patru caractere:

```
mysql> SELECT nume, prenume FROM presedinte
    -> WHERE nume LIKE "____";
```

nume	prenume
Polk	James K.
Taft	William H.
Ford	Gerald R.
Bush	George W.

De asemenea, MySQL furnizează o altă metodă de stabilire a corespondenței cu un model, în funcție de expresiile regulate extinse. Expresiile regulate sunt prezentate în descrierea operatorului `REGEXP` din Anexa C.

Generarea de sumare

Una dintre cele mai utile operații pe care MySQL le poate face pentru dumneavoastră este să rezume mari cantități de date brute și să realizeze sumarul acestora. MySQL devine un aliat puternic atunci când veți învăța să-l folosiți pentru a genera sumare, deoarece aceasta este o activitate extrem de obositoare, consumatoare de timp și supusă la erori atunci când este efectuată manual.

O formă simplă de creare a unui sumar constă în a determina valorile unice prezente într-un set de valori. Folosiți cuvântul cheie `DISTINCT` pentru a elimina rândurile care se repetă dintr-un rezultat. De exemplu, statele distincte în care s-au născut președinții americani se pot găsi astfel:

```
mysql> SELECT DISTINCT stat FROM presedinte ORDER BY stat;
```

stat
AR
CA
GA
IA
IL
KY
MA
MO
NC
NE
NH
NJ
NY
OH
PA
SC
TX
VA
VT

O altă formă de realizare a unui sumar este numărarea, folosind funcția `COUNT()`. Dacă folosiți `COUNT(*)`, aceasta vă indică numărul de rânduri selectat de interogarea dumneavoastră. Dacă interogarea nu are nici o clauză `WHERE`, `COUNT(*)` vă indică numărul rândurilor din tabelul dumneavoastră.

Interogarea următoare vă indică numărul tuturor președinților americani:

```
mysql> SELECT COUNT(*) FROM presedinte;
```

COUNT(*)
41

Dacă o interogare are o clauză `WHERE`, `COUNT(*)` indică numărul de rânduri pe care îl selectează clauza. Această interogare vă arată numărul de chestionare pe care le-ați dat elevilor dumneavoastră până acum:


```
mysql> SELECT COUNT(*) FROM eveniment WHERE type = 'C';
```

COUNT(*)
4

Funcția `COUNT(*)` numără fiecare rând selectat. Prin contrast, `COUNT(nume_coloana)` numără numai valorile diferite de `NULL`. Următoarea interogare demonstrează aceste diferențe:

```
mysql> SELECT COUNT(*),COUNT(sufix),COUNT(data_deces)
    -> FROM presedinte;
```

COUNT(*)	COUNT(sufix)	COUNT(data_deces)
41	1	36

Această interogare arată că în total au fost 41 de președinți, că numele unui singur președinte are un sufix, precum și că majoritatea președinților nu mai sunt în viață.

Începând de la MySQL 3.23.2, puteți combina `COUNT(*)` cu `DISTINCT` pentru a determina numărul de valori distincte dintr-un rezultat. De exemplu, pentru a număra statele distincte în care s-au născut președinții americani, procedați astfel:

```
mysql> SELECT COUNT(DISTINCT stat) FROM presedinte;
```

COUNT(DISTINCT stat)
19

Puteți împărți numărătorile în valori individuale în coloana de sumar. De exemplu, puteți cunoaște numărul total de elevi de la cursul dumneavoastră ca rezultat al rulării următoarei interogări:

```
mysql> SELECT COUNT(*) FROM elev;
```

COUNT(*)
31

Dar câți dintre elevi sunt băieți și câte sunt fete? O modalitate de a afla această informație este de a cere o numărare separată pentru fiecare sex:

```
mysql> SELECT COUNT(*) FROM elev WHERE sex='f';
```

COUNT(*)
15

```
mysql> SELECT COUNT(*) FROM elev WHERE sex='m';
```

COUNT(*)
16

Totuși, deși această metodă funcționează, este obositoare și nu foarte bine adaptată pentru coloanele care pot conține numeroase valori distincte. Să ne gândim cum putem determina în acest mod numărul președinților născuți în fiecare stat. Va trebui să aflați care sunt statele reprezentate, pentru a nu omite nici unul (instrucțiunea `SELECT DIS-`

TINCT stat FROM presedinte), apoi rulați o interogare SELECT COUNT(*) pentru fiecare stat. Categoric, nu doriți să faceți acest lucru.

Din fericire, MySQL poate număra, folosind o singură interogare, de câte ori apare fiecare valoare distinctă într-o coloană. Pentru lista noastră cu elevi, putem număra băieții și fetele astfel:

```
mysql> SELECT sex, COUNT(*) FROM elev GROUP BY sex;
```

sex	COUNT(*)
F	15
M	16

Aceeași formă de interogare indică numărul de președinți născuți în fiecare stat:

```
mysql> SELECT stat, COUNT(*) FROM presedinte GROUP BY stat;
```

stat	COUNT(*)
AR	1
CA	1
GA	1
IA	1
IL	1
KY	1
MA	4
MO	1
NC	2
NE	1
NH	1
NJ	1
NY	4
OH	7
PA	1
SC	1
TX	2
VA	8
VT	2

Când numărați valorile în acest mod, clauza GROUP BY este necesară, deoarece indică programului MySQL modul de grupare a valorilor înainte de a le număra. Dacă omiteți această clauză, veți primi un mesaj de eroare.

Utilizarea funcției COUNT(*) cu GROUP BY pentru numărarea valorilor are un număr de avantaje în raport cu numărarea individuală a aparițiilor fiecărei valori distincte din coloană:

- Nu trebuie să știți dinainte care sunt valorile prezente în coloana al cărei conținut îl rezumați.
- Aveți nevoie de o singură interogare, nu de mai multe interogări.
- Obțineți toate rezultatele într-o singură interogare, deci puteți sorta datele de ieșire.

Primele două avantaje sunt importante pentru exprimarea cu mai mare ușurință a interogărilor. Cel de-al treilea avantaj este important deoarece vă permite o flexibilitate în afișarea rezultatelor.

Când folosiți o clauză GROUP BY, rezultatele sunt sortate în coloanele în funcție de care realizați gruparea, dar puteți folosi ORDER BY pentru a sorta într-o altă ordine. De exemplu, dacă doriți ca numerele de președinți să fie grupate în funcție de statul de naștere, dar să fie și sortate în ordinea descrescătoare a celor mai bine reprezentate state, puteți folosi o clauză ORDER BY după cum urmează:

```
mysql> SELECT stat, COUNT(*) AS numar FROM presedinte
    -> GROUP BY stat ORDER BY numar DESC;
```

stat	numar
VA	8
OH	7
MA	4
NY	4
NC	2
VT	2
TX	2
SC	1
NH	1
PA	1
KY	1
NJ	1
IA	1
MO	1
CA	1
NE	1
GA	1
IL	1
AR	1

Când coloana după care doriți să sortați este determinată prin calcul, puteți da coloanei un alias și puteți face referire la alias în clauza ORDER BY. Interogarea anterioară demonstrează acest fapt; coloana COUNT(*) este denumită numar. O altă modalitate de a face referire la o asemenea coloană este prin poziția sa în datele de ieșire. Interogarea anterioară ar fi putut fi scrisă și astfel:

```
SELECT state, COUNT(*) FROM presedinte
GROUP BY stat ORDER BY 2 DESC
```

Nu cred că desemnarea unei coloane prin poziția acesteia este foarte sugestivă. Dacă adăugați, eliminați sau re-ordonați coloanele cu datele de ieșire, trebuie să vă amintiți să verificați clauza ORDER BY și să corectați numărul coloanei, dacă acesta s-a modificat. Aliasurile nu ridică această problemă.

Dacă doriți să folosiți GROUP BY cu o coloană care conține valori calculate, trebuie să faceți referire la acea coloană folosind un alias sau poziția coloanei, așa cum se procedează în cazul clauzei ORDER BY. Următoarea interogare determină numărul președinților care s-au născut în fiecare lună a anului:

```
mysql> SELECT MONTH(data_nastere) as Luna,
    -> MONTHNAME(data_nastere) AS Nume, COUNT(*) AS numar
    -> FROM presedinte GROUP BY Nume ORDER BY Luna;
```

Luna	Nume	numar
1	Ianuarie	4
2	Februarie	4
3	Martie	4
4	Aprilie	4
5	Mai	2
6	Iunie	1
7	Iulie	3
8	August	4
9	Septembrie	1
10	Octombrie	6
11	Noiembrie	5
12	Decembrie	3

Folosind pozițiile coloanelor, interogarea poate fi scrisă astfel:

```
SELECT MONTH(data_nastere), MONTHNAME(data_nastere), COUNT(*)
FROM presedinte GROUP BY 2 ORDER BY 1;
```

COUNT() poate fi combinată cu `ORDER BY` și `LIMIT` pentru a găsi, de exemplu, cele mai bine reprezentate patru state din tabelul `presedinte`:

```
mysql> SELECT stat, COUNT(*) AS numar FROM presedinte
    -> GROUP BY stat ORDER BY numar DESC LIMIT 4;
```

stat	numar
VA	8
OH	7
MA	4
NY	4

Dacă nu doriți să limitați datele de ieșire ale interogării folosind o clauză `LIMIT`, ci prin căutarea anumitor valori ale funcției `COUNT()`, utilizați o clauză `HAVING`. Interogarea următoare vă va indica numărul de state reprezentate prin doi sau mai mulți președinți:

```
mysql> SELECT stat, COUNT(*) AS numar FROM presedinte
    -> GROUP BY stat HAVING numar > 1 ORDER BY numar DESC;
```

stat	numar
VA	8
OH	7
MA	4
NY	4
NC	2
VT	2
TX	2

În general, acesta este tipul de interogare care trebuie executat atunci când doriți să găsiți într-o coloană valori care se repetă.

`HAVING` este similar cu `WHERE`, dar se aplică după ce au fost selectate rezultatele interogării și este folosit pentru a reduce domeniul rezultatelor trimise efectiv de server clientului.

Există funcții de sumar altele decât `COUNT()`. Funcțiile `MIN()`, `MAX()`, `SUM()` și `AVG()` sunt utile pentru determinarea valorii minime, maxime, totale și medii dintr-o coloană. Le puteți chiar

folosi pe toate simultan. Următoarea interogare prezintă diferite caracteristici numerice pentru fiecare chestionar și test pe care l-ați dat. De asemenea, indică numărul de punctaje incluse în calculul fiecărei valori. (Unii elevi au fost absenți și, ca atare, nu au fost numărați.)

```
mysql> SELECT
    -> eveniment_id,
    -> MIN(puncte) AS minim,
    -> MAX(puncte) AS maxim,
    -> MAX(puncte)-MIN(puncte)+1 AS domeniu,
    -> SUM(puncte) AS total,
    -> AVG(puncte) AS medie,
    -> COUNT(puncte) AS numar,
    -> FROM puncte
    -> GROUP BY eveniment_id;
```

eveniment_id	minim	maxim	domeniu	total	medie	numar
1	9	20	12	439	15.1379	29
2	8	19	12	425	14.1667	30
3	60	97	38	2425	78.2258	31
4	7	20	14	379	14.0370	27
5	8	20	13	383	14.1852	27
6	62	100	39	2325	80.1724	29

Aceste informații pot deveni mai semnificative dacă ați ști, desigur, dacă valorile respective au fost obținute la punctaje sau teste. Totuși, pentru a obține acele informații, trebuie să consultăm și tabelul `eveniment`; vom reveni la această interogare în secțiunea „Regăsirea informațiilor din mai multe tabele".

Funcțiile de sumar sunt distractive, deoarece sunt foarte puternice, dar vă puteți lăsa foarte ușor „dus de val". Priviți această interogare:

```
mysql> SELECT
    -> stat AS Stat,
    -> AVG((TO_DAYS(data_deces)-TO_DAYS(data_nastere))/365) AS Varsta
    -> FROM presedinte WHERE data_deces IS NOT NULL
    -> GROUP BY state ORDER BY Varsta;
```

stat	varsta
KY	56.208219
VT	58.852055
NC	60.141096
OH	62.866145
NH	64.917808
NY	69.342466
NJ	71.315068

stat	varsta
TX	71.476712
MA	72.642009
VA	72.822945
PA	77.158904
SC	78.284932
CA	81.336986
MO	88.693151
IA	90.254795

Interogarea selectează președinții care au decedat, îi grupează în funcție de data nașterii, le calculează vârsta la data decesului, calculează vârsta medie (per stat) și apoi sortează rezultatele după vârsta medie. Cu alte cuvinte, interogarea determină, pentru președinții care nu mai sunt în viață, vârsta medie la data decesului în funcție de statul de naștere.

Şi ce înseamnă asta? Înseamnă numai că puteţi scrie interogarea. Evident, nu înseamnă că interogarea merită scrisă. Nu toate operaţiile pe care le puteţi aplica unei baze de date sunt la fel de semnificative; totuşi, uneori oamenii devin extrem de fericiţi când află ce anume pot face cu baza lor de date. Astfel se poate explica apariţia unui număr tot mai mare de statistici din ce în ce mai ezoterice (şi mai lipsite de sens) cu ocazia evenimentelor sportive televizate din Statele Unite în ultimii ani. Statisticienii sportivi îşi pot folosi bazele de date pentru a determina tot ce doriţi să ştiţi despre o echipă, dar şi tot ce *nu* doriţi să ştiţi. Chiar este atât de important să ştiţi care fundaş din Divizia A (*mutatis mutandis* pentru realitatea românească – N.T.) are la activ cele mai multe intercepţii în careul de 16 metri atunci când echipa lui conduce cu mai mult de două goluri diferenţă, când până la sfârşitul reprizei a doua sunt mai puţin de 15 minute şi antrenorul încă n-a făcut cele trei schimbări regulamentare?

Regăsirea informaţiilor din mai multe tabele

Interogările pe care le-am scris până acum au extras date dintr-un singur tabel. Acum ajungem la partea interesantă. Am mai spus că puterea unui SGBD relaţional rezidă în capacitatea sa de a corela un aspect cu altul, deoarece acest fapt vă permite să combinaţi informaţii din mai multe tabele pentru a găsi răspunsul la întrebări la care nu se poate răspunde consultând un singur tabel. Această secţiune vă prezintă modul de scriere a interogărilor de regăsire a informaţiilor din mai multe tabele.

Când selectaţi informaţii din mai multe tabele, efectuaţi o operaţie denumită *unire*. Aceasta deoarece produceţi un rezultat prin unirea informaţiilor dintr-un tabel cu informaţiile din celălalt tabel, operaţie care are loc prin coroborarea înregistrărilor din mai multe tabele care conţin valori comune.

Să parcurgem un exemplu. Anterior, în secţiunea „Tabele pentru proiectul de evidenţă a rezultatelor şcolare", a fost prezentată, fără nici o explicaţie, o interogare pentru regăsirea punctajelor de la chestionare sau teste date la o anumită dată. Acum este momentul să dăm explicaţia. De fapt, interogarea implică o unire pe trei căi, deci o vom descrie în două etape.

În prima etapă, construim o interogare pentru selectarea punctajelor obţinute la o anumită dată, după cum se poate vedea mai jos:

```
mysql> SELECT elev_id, data, puncte, tip
    -> FROM eveniment, puncte
    -> WHERE data = "1999-09-23"
    -> AND eveniment.eveniment_id = puncte.eveniment_id;
```

elev_id	data	puncte	tip
1	1999-09-23	15	C
2	1999-09-23	12	C
3	1999-09-23	11	C
5	1999-09-23	13	C
6	1999-09-23	18	C
...			

Această interogare funcţionează prin găsirea înregistrării evenimentului cu data precizată şi apoi foloseşte identificatorul de eveniment din acea înregistrare pentru a localiza punctaje cu acelaşi identificator de eveniment. Pentru fiecare combinaţie corespunzătoare de înregistrare de eveniment şi înregistrare de punctaj, sunt afişate identificatorul elevului, punctajul, data şi tipul de eveniment.

Interogarea diferă de cele scrise până acum din două puncte de vedere importante:

- Clauza `FROM` denumeşte mai multe tabele, deoarece datele sunt regăsite din mai multe tabele:

  ```
  FROM eveniment, puncte
  ```

- Clauza `WHERE` precizează că tabelele `eveniment` şi `puncte` sunt unite prin stabilirea unei echivalenţe între valorile coloanei `eveniment_id` din fiecare tabel:

  ```
  WHERE ... eveniment.eveniment_id = puncte.eveniment_id
  ```

Observaţi modul în care se face referire la coloane, folosindu-se sintaxa `nume_tabel.nume_coloana`, astfel încât MySQL să ştie care sunt tabelele în chestiune. (Coloana `eveniment_id` apare în ambele tabele, deci apare o situaţie echivocă dacă o folosim fără un nume de tabel pentru a o identifica.)

Celelalte coloane din interogare (`data`, `puncte`, `tip`) pot fi folosite fără un element de calificare de tip nume de tabel, deoarece apar într-un singur tabel şi, ca atare, sunt lipsite de echivoc. Totuşi, eu în general prefer să calific fiecare coloană dintr-o unire, pentru a preciza mai clar tabelul din care face parte coloana. Într-o formă complet determinată, interogarea se prezintă astfel:

```
SELECT puncte.elev_id, eveniment.data, puncte.puncte, eveniment.tip
FROM eveniment, puncte
WHERE eveniment.data = "1999-09-23"
AND eveniment.eveniment_id = puncte.eveniment_id
```

De acum încolo, voi folosi forma complet determinată.

În cea de-a doua etapă, vom finaliza interogarea folosind tabelul `elev` pentru a afişa numele elevilor. (Datele de ieşire ale interogării din prima etapă ne prezintă câmpul `elev_id`, dar numele sunt mai semnificative.) Afişarea numelor este realizată folosind faptul că atât tabelul `puncte`, cât şi tabelul `elev` conţin coloana `elev_id`, ceea ce permite corelarea înregistrărilor din aceste tabele. Interogarea rezultantă se prezintă astfel:

```
mysql> SELECT elev.nume, eveniment.data, puncte.puncte,
    -> eveniment.tip FROM eveniment, puncte, elev
    -> WHERE eveniment.data = "1999-09-23"
    -> AND eveniment.eveniment_id = puncte.eveniment_id
    -> AND puncte.eveniment_id = elev.elev_id;
```

nume	data	puncte	tip
Megan	1999-09-23	15	C
Joseph	1999-09-23	12	C
Kyle	1999-09-23	11	C
Abby	1999-09-23	13	C
Nathan	1999-09-23	18	C
...			

Această interogare diferă de precedenta din următoarele puncte de vedere:

- Tabelul `elev` este adăugat la clauza `FROM` deoarece este folosit alături de tabelele `eveniment` și `puncte`.
- Coloana `elev_id` este acum echivocă, deci trebuie determinată sub forma `puncte.elev_id` sau `elev.elev_id`, pentru a preciza clar care este tabelul ce urmează a fi folosit. (Acest fapt este valabil chiar dacă preferați să scrieți uniri care nu precizează complet fiecare referință la o coloană.)
- Clauza `WHERE` are un termen suplimentar, care arată că înregistrările din tabelul `puncte` sunt coroborate cu înregistrările din tabelul `elev`, în funcție de identificatorul fiecărui elev:

  ```
  WHERE ... puncte.elev_id = elev.elev_id
  ```
- Interogarea afișează numele elevului, nu identificatorul său. (Bineînțeles că le puteți afișa pe ambele, dacă doriți.)

Cu această interogare, puteți introduce orice dată și obțineți punctajele de la data respectivă, alături de numele elevilor și de tipul punctajului. Nu trebuie să știți nimic despre identificatorii de elev sau de eveniment. MySQL se ocupă de determinarea valorilor relevante ale identificatorilor și de utilizarea lor pentru coroborarea automată a informațiilor din rândurile tabelelor.

O altă sarcină pe care o incumbă proiectul de evidență a rezultatelor școlare este totalizarea absențelor elevilor. Absențele sunt înregistrate în tabelul `absente`, în funcție de identificatorul elevului și de dată. Pentru a obține numele elevilor (nu doar identificatorii lor) trebuie să unim tabelul `absente` cu tabelul `elev`, în funcție de valoarea din coloana `elev_id`. Interogarea următoare afișează numărul de identificare a elevului și numele acestuia, alături de numărul absențelor:

```
mysql> SELECT elev.elev_id, elev.nume,
    -> COUNT(absente.data) AS absente
    -> FROM elev, absente
    -> WHERE elev.elev_id = absente.elev_id
    -> GROUP BY elev.elev_id;
```

elev_id	nume	absente
3	Kyle	1
5	Abby	1
10	Peter	2
17	Will	1
20	Avery	1

Notă: Deși am furnizat un calificator în clauza `GROUP BY`, nu este absolut necesar pentru această interogare. `GROUP BY` se referă la coloanele din lista de selecție (din primele două linii ale interogării). Aici există o singură coloană denumită `elev_id`, deci MySQL cunoaște coloana la care vă referiți. Acest fapt este de asemenea valabil pentru coloanele specificate în clauzele `ORDER BY`.

Datele de ieșire generate de interogare sunt acceptabile dacă dorim să știm numai care sunt elevii cu absențe. Dar, dacă trimitem această listă la secretariatul școlii, cei de acolo vor spune: „Dar ceilalți elevi? Vrem o valoare pentru fiecare elev." Aceasta este o problemă ușor diferită. Înseamnă că dorim să cunoaștem numărul absențelor, chiar și pen-

tru elevii care nu au avut nici o absență. Deoarece întrebarea este diferită, și interogarea este diferită.

Pentru a răspunde la întrebare, putem folosi `LEFT JOIN` în loc de a asocia identificatorul de elev în clauza `WHERE`. `LEFT JOIN` indică programului MySQL să genereze un rând de date de ieșire pentru fiecare rând selectat din tabelul denumit primul în cadrul unirii (adică tabelul precizat în stânga cuvintelor cheie `LEFT JOIN`). Precizând mai întâi tabelul `elev`, vom primi date de ieșire pentru fiecare elev, chiar și pentru cei care nu sunt reprezentați în tabelul `absente`. Interogarea se prezintă astfel:

```
mysql> SELECT elev.elev_id, elev.nume,
    -> COUNT(absente.date) AS absente
    -> FROM elev LEFT JOIN absente
    -> ON elev.elev_id = absente.elev_id
    -> GROUP BY elev.elev_id;
```

elev_id	nume	absente
1	Megan	0
2	Joseph	0
3	Kyle	1
4	Katie	0
5	Abby	1
6	Nathan	0
7	Liesl	0
...		

Anterior, în secțiunea „Generarea de sumare" am rulat o interogare care producea o caracterizare numerică a datelor din tabelul `puncte`. Datele de ieșire ale acelei interogări afișau identificatorul de eveniment, dar nu includeau data la care a fost obținut punctajul sau tipul acestuia, deoarece atunci nu știam cum să unim tabelul `puncte` cu tabelul `eveniment`, pentru a obține datele și tipurile punctajelor. Acum știm. Interogarea următoare este similară celei pe care am executat-o anterior, dar prezintă data și tipul punctajelor, nu numai identificatorul numeric de eveniment:

```
mysql> SELECT
    -> eveniment.data,eveniment.tip
    -> MIN(puncte.puncte) AS minim,
    -> MAX(puncte.puncte) AS maxim,
    -> MAX(puncte.puncte)-MIN(puncte.puncte)+1 AS domeniu,
    -> SUM(puncte.puncte) AS total,
    -> AVG(puncte.puncte) AS medie,
    -> COUNT(puncte.puncte) AS numar,
    -> FROM puncte, eveniment
    -> WHERE puncte.eveniment_id = eveniment.eveniment_id
    -> GROUP BY eveniment.data;
```

data	tip	minim	maxim	domeniu	total	medie	numar
1999-09-03	C	9	20	12	439	15.1379	29
1999-09-06	C	8	19	12	425	14.1667	30
1999-09-09	T	60	97	38	2425	78.2258	31
1999-09-16	C	7	20	14	379	14.0370	27
1999-09-23	C	8	20	13	383	14.1852	27
1999-10-01	T	62	100	39	2325	80.1724	29

Puteți folosi funcții precum `COUNT()` sau `AVG()` pentru a genera un sumar pe coloane multiple, chiar dacă aceste coloane provin de la tabele diferite. Interogarea următoare determină numărul de punctaje și punctajul mediu pentru fiecare combinație între data evenimentului și sexul elevului:

```
mysql> SELECT eveniment.data, elev.sex,
    -> COUNT(puncte) AS numar, AVG(puncte) AS medie
    -> FROM eveniment, puncte, elev
    -> WHERE eveniment.eveniment_id = puncte.elev_id
    -> AND puncte.elev_id = elev.elev_id
    -> GROUP BY eveniment.data, elev.sex;
```

data	sex	numar	medie
1999-09-03	F	14	14.6429
1999-09-03	M	15	15.6000
1999-09-06	F	14	14.7143
1999-09-06	M	16	13.6875
1999-09-09	F	15	77.4000
1999-09-09	M	16	79.0000
1999-09-16	F	13	15.3077
1999-09-16	M	14	12.8571
1999-09-23	F	12	14.0833
1999-09-23	M	15	14.2667
1999-10-01	F	14	77.7857
1999-10-01	M	15	82.4000

Putem folosi o interogare similară pentru a efectua una dintre sarcinile aferente proiectului de evidență a rezultatelor școlare: calculul punctajului total al fiecărui elev la sfârșitul semestrului. Interogarea se prezintă astfel:

```
SELECT elev.elev_id, elev.nume
SUM(puncte.puncte) AS total, COUNT(puncte.puncte) AS n
FROM eveniment, puncte, elev
WHERE eveniment.eveniment_id = puncte.eveniment_id
AND puncte.elev_id = elev.elev_id
GROUP BY puncte.elev_id
ORDER BY total
```

Nu există nici o cerință conform căreia o unire trebuie efectuată folosindu-se două tabele diferite. La început poate părea ciudat, dar puteți uni un tabel cu el însuși. De exemplu, puteți determina dacă au existat președinți născuți în același oraș comparând locul nașterii fiecărui președinte cu locul nașterii tuturor celorlalți președinți:


```
mysql> SELECT p1.nume, p1.prenume, p1.oras, p1.stat
    -> FROM presedinte AS p1, presedinte AS p2
    -> WHERE p1.oras = p2.oras AND p1.stat = p2.stat
    -> AND (p1.nume != p2.nume OR p1.prenume != p2.prenume)
    -> ORDER stat, oras, nume;
```

nume	prenume	oras	stat
Adams	John Quincy	Braintree	MA
Adams	John	Braintree	MA

Există două aspecte ciudate ale acestei interogări:

- Trebuie să facem referire la două instanțe ale aceluiași tabel, deci vom crea aliasuri ale tabelului (`p1`, `p2`) și le vom folosi pentru a explicita referirile la coloanele tabelului.
- Înregistrarea fiecărui președinte este echivalentă cu ea însăși, dar nu dorim să vedem acest lucru în datele de ieșire. Cea de-a doua linie a clauzei `WHERE` interzice corespondența unei înregistrări cu ea însăși, asigurând faptul că înregistrările comparate aparțin unor președinți diferiți.

O interogare similară găsește președinții care s-au născut în aceeași zi. Datele de naștere nu pot fi comparate direct, deoarece astfel s-ar „pierde" președinții care s-au născut în ani diferiți. În schimb, vom folosi `MONTH()` și `DAYOFMONTH()` pentru a compara ziua și luna datei de naștere:

```
mysql> SELECT p1.nume, p1.prenume, p1.data_nastere
    -> FROM presedinte AS p1, presedinte AS p2
    -> WHERE MONTH(p1.data_nastere) = MONTH(p2.data_nastere)
    -> AND DAYOFMONTH(p1.data_nastere) =
    -> DAYOFMONTH(p2.data_nastere)
    -> AND (p1.nume != p2.nume OR p1.prenume != p2.prenume)
    -> ORDER BY p1.nume;
```

nume	prenume	data_nastere
Harding	Warren G.	1865-11-02
Polk	James K.	1795-11-02

Folosind funcția `DAYOFYEAR()` în locul combinației între funcțiile `MONTH()` și `DAYOFMONTH()` s-ar obține o interogare mai simplă, dar s-ar produce rezultate incorecte atunci când se compară datele din anii bisecți cu datele din anii cu 365 de zile.

Unirile efectuate până acum au combinat informații din tabele între care există o oarecare relație logică semnificativă, dar numai dumneavoastră cunoașteți semnificația acelei relații. MySQL nu știe (sau nu-i pasă) dacă tabelele unite au sau nu vreo legătură unul cu altul. De exemplu, puteți uni tabelul `eveniment` cu tabelul `presedinte`, pentru a afla dacă ați dat sau nu chestionare sau teste de ziua de naștere a unui președinte:

```
mysql> SELECT presedinte.nume, presedinte.prenume,
    -> presedinte.data_nastere, eveniment.tip
    -> FROM presedinte, eveniment
    -> WHERE MONTH(presedinte.data_nastere) =
```

```
    -> MONTH(eveniment.data)
    -> AND DAYOFMONTH(presedinte.data_nastere) =
    -> DAYOFMONTH(eveniment.data);
```

nume	prenume	data_nastere	tip
Carter	James E	1924-10-01	T

Se pare că aţi dat un test de ziua de naştere a unui preşedinte. Şi ce dacă? Aceasta înseamnă că MySQL va fi foarte bucuros să scoată la iveală rezultate, indiferent dacă ele sunt logice sau nu. Numai pentru că folosiţi un calculator nu înseamnă obligatoriu că rezultatele unei interogări sunt utile sau că merită osteneala. Din fericire sau din păcate, nu suntem scutiţi de a şti ce trebuie să facem.

Ştergerea sau actualizarea înregistrărilor existente

Uneori, doriţi să vă descotorosiţi de înregistrări sau doriţi să le modificaţi conţinutul. Instrucţiunile `DELETE` şi `UPDATE` vă permit aceste operaţii.

Instrucţiunea `DELETE` are următoarea formă:

```
DELETE FROM tabel_nume WHERE inregistrari care trebuie sterse
```

Clauza `WHERE` indică înregistrările care trebuie şterse şi este opţională dar, dacă o omiteţi, vor fi şterse toate înregistrările. Aceasta înseamnă că instrucţiunea `DELETE` cea mai simplă este şi cea mai periculoasă:

```
DELETE FROM nume_tabel
```

Această interogare şterge în totalitate conţinutul tabelului. *Fiţi atent la interogarea respectivă!*

Pentru a şterge anumite înregistrări, folosiţi clauza `WHERE` pentru a selecta înregistrările care vă interesează. Aceasta este similară clauzei `WHERE` dintr-o instrucţiune `SELECT`. De exemplu, pentru a şterge din tabelul `presedinte` pe toţi preşedinţii născuţi în statul Ohio, folosiţi această interogare:

```
mysql> DELETE FROM presedinte WHERE stat="OH";
Query OK, 7 rows affected (0.00 sec)
```

Un dezavantaj al clauzei `WHERE` pentru instrucţiunea `DELETE` este acela că puteţi face referire numai la coloanele tabelului din care ştergeţi înregistrările.

Înainte de a emite o instrucţiune `DELETE`, deseori este bine să testaţi clauza `WHERE` folosind-o cu o instrucţiune `SELECT`, pentru a vă asigura că veţi şterge înregistrările pe care doriţi să le ştergeţi (şi *numai* acele înregistrări). Să presupunem că doriţi să ştergeţi înregistrarea pentru Teddy Roosevelt. Ar fi bună următoarea interogare?

```
DELETE FROM presedinte WHERE nume="Roosevelt"
```

Da, în sensul că ar şterge înregistrarea pe care o aveţi în vedere. Nu, în sensul că ar şterge şi înregistrarea lui Franklin Roosevelt. Cel mai bine este să verificaţi în prealabil clauza `WHERE`, astfel:

```
mysql> SELECT nume, prenume FROM presedinte
    -> WHERE nume="Roosevelt";
```


prenume	nume
Roosevelt	Theodore
Roosevelt	Franklin D.

Din acest exemplu se deduce necesitatea de a fi mai concret:

```
mysql> SELECT nume, prenume FROM presedinte
    -> WHERE nume="Roosevelt" AND prenume="Theodore";
```

prenume	nume
Roosevelt	Theodore

Acum ştim care este clauza `WHERE` corectă pentru selectarea înregistrării dorite, deci interogarea `DELETE` poate fi construită corect:

```
mysql> DELETE FROM presedinte
    -> WHERE nume="Roosevelt" AND prenume="Theodore";
```

Cam multă muncă pentru a şterge o înregistrare, nu-i aşa? Paza bună trece primejdia rea! (Aceasta este una din situaţiile în care veţi dori să reduceţi la minimum numărul caracterelor introduse de la tastatură, prin utilizarea tehnicilor de copiere şi lipire, respectiv de editare a liniilor de intrare. Vezi secţiunea „Sugestii pentru interacţiunea cu `mysql`" pentru mai multe informaţii.)

Pentru modificarea înregistrărilor existente, folosiţi `UPDATE`, care are următoarea formă:

```
UPDATE nume_tabel SET coloane de modificat
WHERE inregistrari de actualizat
```

Clauza `WHERE` se foloseşte ca în cazul instrucţiunii `DELETE` şi este opţională deci, dacă nu o specificaţi, *vor fi actualizate toate înregistrările din tabel.* Interogarea următoare transformă numele tuturor elevilor dumneavoastră în „George":

```
mysql> UPDATE elev SET nume="George";
```

Evident, trebuie să fiţi atent cu înregistrări de acest gen.

De obicei, veţi fi mai concret în ceea ce priveşte înregistrările pe care le actualizaţi. Să presupunem că aţi adăugat recent un membru al Ligii istorice, dar aţi completat numai câteva coloane din rubrica sa:

```
mysql> INSERT membru (nume,prenume)
    -> VALUES('Ionescu','Vasile');
```

Apoi, vă daţi seama că aţi uitat să menţionaţi data plăţii cotizaţiei. Puteţi remedia acest neajuns după cum urmează:

```
mysql> UPDATE membru
    -> SET data_expirare='2001-7-20'
    -> WHERE nume='Ionescu' AND prenume='Vasile'
```

Puteţi actualiza mai multe coloane simultan. Instrucţiunea următoare actualizează adresa poştală şi de e-mail a lui Vasile:

```
mysql> UPDATE membru
    -> SET email ='ivasile@yahoo.com',
    -> strada='123 Str. Lunga',oras='Nicaieri',
```

```
    -> stat='NY',cod_postal='01003'
    -> WHERE nume='Ionescu' AND prenume='Vasile';
```

De asemenea, puteți „anula" valoarea unei coloane atribuindu-i acesteia valoarea NULL (presupunând că acea coloană acceptă valori NULL). Dacă la un anumit moment în viitor Vasile se decide să plătească acea cotizație mare care îi permite să devină membru „pe viață", îi puteți modifica înregistrarea în mod adecvat atribuind datei de expirare valoarea NULL (în sensul că „nu expiră niciodată"):

```
mysql> UPDATE membru
    -> SET data_expirare=NULL
    -> WHERE nume='Ionescu' AND prenume='Vasile';
```

În cazul instrucțiunii UPDATE, ca și pentru DELETE, este bine să testați o clauză WHERE folosind o instrucțiune SELECT, pentru a vă asigura că alegeți înregistrările corecte pentru actualizare. În cazul în care criteriile dumneavoastră de selecție sunt prea înguste sau prea largi, veți actualiza prea puține, respectiv prea multe înregistrări.

Dacă ați încercat interogările din această secțiune, veți fi șters și modificat înregistrări din tabelele bazei de date samp_db. Înainte de a trece la secțiunea următoare, trebuie să anulați aceste modificări. Faceți aceasta prin reîncărcarea tabelelor folosind instrucțiunile de la sfârșitul secțiunii „Adăugarea de noi înregistrări".

Modificarea structurii tabelelor

Vă mai amintiți când am creat tabelul membru al Ligii istorice fără o coloană a identificatorului numeric al membrilor, ca pretext pentru a folosi instrucțiunea ALTER TABLE? Este momentul să folosim această instrucțiune. Cu ajutorul instrucțiunii ALTER TABLE, puteți redenumi tabele, adăuga sau șterge coloane, modifica tipuri de coloane și altele. Singurul exemplu pe care îl voi prezenta aici este modalitatea de a adăuga o coloană nouă. Vezi capitolul 3 pentru mai multe detalii despre posibilitățile instrucțiunii ALTER TABLE.

Considerentul principal la adăugarea unei coloane cu identificatorul numeric al membrilor la tabelul membru este acela că valorile trebuie să fie unice, pentru a se evita confuzia între rubrici. Aici este utilă o coloană AUTO_INCREMENT, deoarece putem permite programului MySQL să genereze automat numere unice atunci când adăugăm membri noi. Într-o instrucțiune CREATE TABLE, specificația pentru o asemenea coloană ar arăta astfel:

```
membru_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
```

Pentru instrucțiunea ALTER TABLE, sintaxa este similară. Rulați această interogare pentru a adăuga coloana:

```
mysql> ALTER TABLE membru
    -> ADD membru_id INT UNSIGNED NOT NULL AUTO_INCREMENT
    -> PRIMARY KEY;
```

Acum, când avem o nouă coloană pentru memorarea identificatorilor numerici ai membrilor, cum repartizăm numere înregistrărilor existente din tabelul membru? Simplu; MySQL s-a ocupat deja de asta! Când adăugați o coloană la un tabel, MySQL inițializează valorile coloanelor cu valoarea prestabilită. În cazul unei coloane AUTO_INCREMENT, aceasta determină generarea unui nou număr din secvență pentru fiecare rând.

Sugestii pentru interacțiunea cu mysql

Această secțiune discută despre modul de a interacționa cu programul client mysql într-o manieră mai eficientă și cu mai puțin text introdus de la tastatură. Este descris un mod mai simplu de conectare la server și metode de introducere a interogărilor fără a o tasta pe fiecare de la început până la sfârșit.

Simplificarea procesului de conexiune

Probabil că trebuie să specificați parametri de conexiune precum numele gazdei, numele de utilizator sau parola atunci când invocați mysql. Pentru simpla rulare a unui program trebuie introdusă o cantitate serioasă de text, lucru care devine foarte repede obositor. Există numeroase metode pentru simplificarea procesului de conectare, prin reducerea la minimum a textului introdus de la tastatură:

- Utilizați un fișier cu opțiuni pentru stocarea parametrilor de conectare.
- Repetați comenzile beneficiind de caracteristicile interpretorului dumneavoastră de memorare a comenzilor (istoric).
- Definiți o scurtătură pentru linia de comandă mysql folosind un alias sau un script de interpretor.

Utilizarea unui fișier cu opțiuni

Începând de la versiunea 3.22, MySQL vă permite să stocați parametri de conexiune într-un fișier cu opțiuni. Apoi, nu mai trebuie să tastați parametri de fiecare dată când rulați mysql; aceștia sunt folosiți ca și cum i-ați fi introdus în linia de comandă. Parametrii sunt de asemenea folosiți și de către alte programe client MySQL, precum mysqlimport. Aceasta înseamnă că un fișier cu opțiuni reduce cantitatea de text introdus atunci când folosiți și aceste programe.

Pentru a folosi fișierul cu opțiuni în vederea specificării parametrilor de conexiune, creați un fișier denumit ~/.my.cnf (adică un fișier denumit .my.cnf în catalogul dumneavoastră de bază). Un fișier cu opțiuni este un fișier cu text simplu, deci îl puteți crea folosind orice editor de texte. Conținutul fișierului se prezintă astfel:

```
[client]
host=gazda_serverului
user=numele_dumneavoastra
password=parola_dumneavoastra
```

Linia [client] indică începutul grupului cu opțiuni pentru programul client; toate liniile care urmează după aceasta sunt linii citite de către programele client MySQL pentru a obține valorile opțiunilor, până la sfârșitul fișierului sau până la începutul unui grup de parametri diferit. Înlocuiți gazda_serverului, numele_dumneavoastra și parola_dumneavoastra cu numele gazdei, numele de utilizator și parola pe care le specificați când vă conectați la server. Pentru mine, fișierul .my.cnf se prezintă astfel

```
[client]
host=pit-viper.snake.net
user=paul
password=secret
```

Singura linie obligatorie este `[client]`. Liniile care definesc valorile parametrilor sunt opționale; le puteți specifica numai pe cele dorite. De exemplu, dacă numele dumneavoastră de utilizator MySQL este același cu numele UNIX de deschidere a sesiunii de lucru, atunci nu trebuie să includeți o linie `user`.

După ce ați creat fișierul `.my.cnf`, stabiliți pentru modul său de acces o valoare restrictivă, pentru a vă asigura că nimeni altcineva nu poate să-l citească:

```
% chmod 600 .my.cnf
```

Sub Windows, conținutul fișierului cu opțiuni este același, deși numele este diferit (`C:\my.cnf`), iar dumneavoastră nu invocați comanda `chmod`.

Deoarece fișierele cu opțiuni au fost adăugate la MySQL decât începând cu versiunea 3.22, nu le puteți folosi cu versiunile anterioare. În particular, nu puteți folosi un fișier cu opțiuni sub Windows la clienții incluși în distribuția shareware a sistemului MySQL, deoarece aceea se bazează pe MySQL 3.21. Fișierele cu opțiuni sunt compatibile cu versiunea Windows înregistrată a sistemului MySQL, sau vă puteți procura programe client mai recente, compatibile cu fișierele cu opțiuni, din situl Web MySQL.

Despre fișierele cu opțiuni se pot obține mai multe informații în Anexa E, „Referință de program MySQL".

Utilizarea istoricului de comenzi al interpretorului

Interpretoare precum `csh`, `tcsh` și `bash` rețin comenzile dumneavoastră într-o listă istoric și vă permit să repetați comenzi din lista respectivă. Dacă folosiți un asemenea interpretor, lista dumneavoastră istoric vă poate ajuta să evitați tastarea unor întregi comenzi. De exemplu, dacă ați invocat programul `mysql` recent, îl puteți reexecuta după cum urmează:

```
% !my
```

Caracterul `!` indică interpretorului să caute în istoricul de comenzi pentru a găsi cea mai recentă comandă care începe cu `my` și să o re-execute ca și cum dumneavoastră ați fi reintrodus-o de la tastatură. De asemenea, unele interpretoare vă permit să vă deplasați în sus și în jos în lista istoric folosind tastele cu săgeți orientate în sus, respectiv în jos (sau poate combinațiile de taste Ctrl-P și Ctrl-N). Puteți selecta în acest mod comanda dorită și apoi apăsați pe Enter pentru a o executa. `tcsh` și `bash` au această caracteristică, posibil existentă și la alte interpretoare. Examinați documentația interpretorului dumneavoastră pentru a afla mai multe detalii despre utilizarea listei istoric.

Utilizarea aliasurilor și a scripturilor de interpretor

Dacă interpretorul dumneavoastră permite atribuirea unui alias, puteți configura scurtături de comenzi care vă permit să invocați o comandă lungă prin tastarea unui nume scurt. De exemplu, în `csh` sau `tcsh`, puteți folosi comanda `alias` pentru a configura un alias denumit `samp_db`, cum este acesta:

```
alias samp_db 'mysql -h pit-viper.snake.net -u -paul -p samp_db'
```

Sintaxa pentru `bash` este ușor diferită:

```
alias samp_db='mysql -h pit-viper.snake.net -u -paul -p samp_db'
```

Definirea unui alias face aceste două comenzi echivalente:

```
samp_db
mysql -h pit-viper.snake.net -u -paul -p samp_db
```


Evident, prima comandă este mai ușor de introdus decât a doua. Pentru ca aliasul să aibă efect de fiecare dată când deschideți sesiunea de lucru, plasați comanda `alias` într-unul din fișierele de pornire ale interpretorului dumneavoastră (de exemplu `.cshrc` pentru `csh` sau `.bash_profile` pentru `bash`).

Un alt tip de scurtătură este crearea unui script de interpretor care execută `mysql` automat, cu opțiunile adecvate. În UNIX, un fișier script echivalent cu aliasul `samp_db` prezentat anterior arată astfel:

```
#! /bin/sh
exec mysql -h pit-viper.snake.net -u -paul -p samp_db
```

Dacă denumesc scriptul `samp_db` și îl transform într-un executabil (folosind comanda `chmod +x samp_db`), pot tasta `samp_db` pentru a rula `mysql` și pentru a mă conecta la baza mea de date.

Sub Windows, în aceleași scopuri se poate folosi un fișier batch, având extensia .bat și conținând comenzi executabile fără intervenția utilizatorului. Denumiți fișierul `samp_db.bat` și inserați în fișier o linie ca aceasta:

```
mysql -h pit-viper.snake.net -u -paul -p samp_db
```

Acest fișier batch poate fi rulat fie tastând `samp_db` la promptul dintr-o consolă DOS, fie executând dublu clic pe pictograma sa din Windows.

Dacă obțineți acces la mai multe baze de date sau vă conectați la mai multe gazde, puteți defini mai multe aliasuri sau scripturi, fiecare din acestea invocând `mysql` cu opțiuni diferite.

Emiterea de interogări cu o cantitate mai mică de text introdus de la tastatură

`mysql` este un program extrem de util pentru interacțiunea cu baza dumneavoastră de date, dar interfața sa este ideală pentru interogări scurte, de un rând. `mysql` nu este interesat dacă o interogare se extinde pe mai multe rânduri, desigur, dar nu este prea amuzant să tastezi interogări lungi. Introducerea unei interogări, chiar și a uneia scurte, nu este nici ea prea distractivă, dacă descoperi că trebuie să o tastezi din nou datorită unei erori de sintaxă.

Există numeroase tehnici pe care le puteți utiliza pentru evitarea unor introduceri și re-introduceri inutile de text:

- Folosiți caracteristica de editare a liniei de intrare din `mysql`.
- Utilizați copierea și lipirea.
- Rulați `mysql` în mod batch.
- Evitați să introduceți instrucțiuni `INSERT`, prin utilizarea datelor existente pentru crearea de noi înregistrări.

Utilizarea editorului liniilor de intrare din `mysql`

În `mysql` este încorporată biblioteca GNU Readline, care permite editarea liniilor de intrare. Puteți manipula linia pe care o introduceți la un moment dat sau puteți relua liniile introduse anterior și le puteți re-introduce, fie ca atare, fie după o modificare ulterioară. Acest lucru este convenabil când introduceți o linie și observați o greșeală de

introducere a textului; puteți reveni în interiorul liniei pentru a corecta problema înainte de a apăsa pe tasta Enter. Dacă introduceți o interogare care conține o greșeală, puteți relua interogarea și o puteți edita pentru a remedia problema, după care o retrimiteți. (Acest lucru este cel mai ușor de făcut dacă tastați întreaga interogare pe o singură linie.)

Unele dintre secvențele de editare pe care le veți găsi utile sunt prezentate în tabelul 1.4; dar există multe comenzi de editare a liniei de intrare în afara celor prezentate în tabel. Puteți găsi o versiune electronică a manualului Readline prin consultarea motorului dumneavoastră preferat de căutare în Internet. Manualul este de asemenea inclus în distribuția Readline, disponibilă în situl Web al proiectului GNU, la adresa `http://www.gnu.org/`.

Tabelul 1.4 Comenzi `mysql` de editare a liniei de intrare

Secvență de taste	Semnificație
Săgeată sus, Ctrl-P	Reluarea liniei anterioare
Săgeată jos, Ctrl-N	Reluarea liniei ulterioare
Săgeată stânga, Ctrl-B	Deplasarea cursorului la stânga (înapoi)
Săgeată dreapta, Ctrl-F	Deplasarea cursorului la dreapta (înainte)
Esc Ctrl-B	Deplasare înapoi cu un cuvânt
Esc Ctrl-F	Deplasare înainte cu un cuvânt
Ctrl-A	Deplasarea cursorului la începutul liniei
Ctrl-E	Deplasarea cursorului la sfârșitul liniei
Ctrl-D	Ștergerea caracterului aflat sub cursor
Delete	Ștergerea caracterului din stânga cursorului
Esc D	Ștergerea cuvântului
Esc Backspace	Ștergerea cuvântului din stânga cursorului
Ctrl-K	Șterge tot, de la cursor la sfârșitul liniei
Ctrl-_	Anulează ultima modificare; se poate repeta

Exemplul următor descrie o utilizare simplă pentru editarea liniei de intrare. Să presupunem că ați introdus această interogare în timp ce utilizați `mysql`:

```
mysql> SHOW COLUMNS FROM persedinte;
```

Dacă observați că în loc de „președinte" ați scris „perședinte", înainte de a apăsa pe tasta Enter, apăsați tasta săgeată stânga sau combinația de taste Ctrl-B de câteva ori pentru a deplasa cursorul la stânga, până când ajunge la litera `s`. Apoi apăsați de două ori pe tasta Delete pentru a șterge caracterele `er`, tastați `re` pentru a remedia eroarea și apăsați pe tasta Enter pentru a emite interogarea. Dacă apăsați pe tasta Enter înainte de a sesiza eroarea, nu este nici o problemă. După ce `mysql` își afișează mesajul de eroare, apăsați pe tasta săgeată sus sau pe combinația de taste Ctrl-P pentru a relua linia, după care o editați așa cum am arătat mai sus.

Editarea liniilor de intrare nu este disponibilă în versiunea Windows a programului `mysql`, dar vă puteți procura distribuția client gratuită `cygwin_32` din situl Web MySQL. Programul `mysqlc` din acea distribuție este similar cu `mysql`, dar înțelege comenzile de editare a liniei de intrare.

Utilizarea copierii și a lipirii pentru emiterea de interogări

Dacă lucrați într-un mediu cu ferestre, puteți salva textul interogărilor care vi se par utile într-un fișier și puteți folosi operațiile de copiere și lipire pentru a emite cu ușurință aceste interogări. Pur și simplu respectați următoarea procedură:

1. Invocați `mysql` într-o fereastră Telnet sau într-o fereastră de consolă DOS.
2. Deschideți fișierul care conține interogările dumneavoastră într-o fereastră document. (De exemplu, eu folosesc BBEdit sub Mac OS, respectiv `vi` într-o fereastră `xterm` sub X Window System în UNIX.)
3. Pentru a executa o interogare stocată în fișierul dumneavoastră, selectați-o și copiați-o. Apoi, comutați în fereastra dumneavoastră Telnet sau în consola DOS și lipiți interogarea în MySQL.

Procedura pare cam stângace descrisă pe hârtie, dar este o modalitate rapidă de a introduce interogări rapid și fără a mai scrie nimic atunci când o folosiți efectiv.

De asemenea, această tehnică vă permite să vă editați interogările în fereastra document, precum și să construiți interogări noi prin copierea și lipirea componentelor interogărilor existente. De exemplu, dacă selectați frecvent rânduri dintr-un anumit tabel, dar doriți să vedeți datele de ieșire sortate în diferite moduri, puteți folosi o listă cu diferite clauze `ORDER BY` în fereastra dumneavoastră document, după care o copiați și o lipiți pe aceea pe care doriți să o utilizați pentru orice interogare specifică.

Puteți folosi copierea și lipirea și în cealaltă direcție (de la Telnet către fișierul dumneavoastră cu interogări). Când introduceți linii în `mysql`, acestea sunt salvate într-un fișier denumit `.mysql_history` din catalogul dumneavoastră de bază. Dacă introduceți manual o interogare pe care doriți să o salvați pentru a o consulta ulterior, părăsiți programul `mysql`, deschideți `.mysql_history` într-un editor, apoi copiați și lipiți interogarea din fișierul `.mysql_history` în fișierul care conține interogarea.

Rularea programului `mysql` în mod batch

Nu este necesară rularea interactivă a programului `mysql`; acest program poate citi un fișier în mod non-interactiv (batch). Acest mod este util pentru interogări pe care le rulați periodic, deoarece nu doriți, evident, să reintroduceți de la tastatură o asemenea interogare de fiecare dată când o rulați. Este mai ușor să inserați o dată interogarea într-un fișier, după care să cereți programului `mysql` să execute conținutul fișierului conform necesităților.

Să presupunem că aveți o interogare care găsește pe membrii Ligii istorice interesați de o anumită perioadă din istoria Statelor Unite, căutând în coloana `interese` a tabelului `membru`. De exemplu, pentru a-i găsi pe membrii interesați în perioada de criză a anilor '30, interogarea trebuie scrisă astfel (rețineți caracterul punct și virgulă de la sfârșitul instrucțiunii, pentru ca `mysql` să poată sesiza locul unde se încheie aceasta):

```
SELECT nume, prenume, email, interese FROM membru
WHERE interese LIKE "%criza%"
ORDER BY nume, prenume;
```

Plasați interogarea într-un fișier denumit `interese.sql`, apoi rulați interogarea furnizând fișierul programului `mysql`, astfel:

```
% mysql samp_db < interese.sql
```

În mod prestabilit, mysql generează date de ieşire într-un format delimitat prin tabulatori atunci când este rulat în mod grup. Dacă doriţi să obţineţi acelaşi format tabular atunci când rulaţi mysql în mod interactiv, folosiţi opţiunea -t:

```
% mysql -t samp_db < interese.sql
```

Dacă doriţi să salvaţi datele de ieşire, redirecţionaţi-le spre un fişier:

```
% mysql -t samp_db < interese.sql > fisier_iesire
```

Pentru a folosi interogarea în scopul de a găsi membri interesaţi de personalitatea lui Thomas Jefferson, puteţi edita fişierul cu interogarea pentru a înlocui criza cu Jefferson, după care rulaţi mysql din nou. Această metodă funcţionează atâta timp cât nu folosiţi interogarea foarte frecvent. Dacă o folosiţi foarte frecvent, atunci este necesară o altă metodă. O modalitate de a spori flexibilitatea interogării este de a o insera într-un script de interpretor care preia un argument din linia de comandă a scriptului şi o foloseşte pentru a modifica textul interogării. Astfel, interogarea devine parametrizată şi vă permite să specificaţi cuvântul cheie care arată domeniul de interes atunci când rulaţi scriptul. Pentru a vedea modul de funcţionare al acestei metode, scrieţi un mic script de interpretor, denumit interese.sh:

```
#! /bin/sh
if [ $# -ne 1 ]; then echo "Rog specificati un cuvant cheie"; exit; fi
mysql -t samp_db <<QUERY_INPUT
SELECT nume, prenume, email, interese FROM membru
WHERE interese LIKE "%$1%"
ORDER BY nume, prenume;
QUERY_INPUT
```

Cea de-a doua linie se asigură că există un cuvânt cheie în linia de comandă; afişează un scurt mesaj şi încheie execuţia programului în caz contrar. Întreg textul cuprins între <<QUERY_INPUT şi linia QUERY_INPUT finală se transformă în date de intrare pentru mysql. În cadrul textului interogării, interpretorul înlocuieşte referinţa la $1 cu cuvântul cheie din linia de comandă. (În scripturile de interpretor, $1, $2... se referă la argumentele comenzii.) Acest fapt determină interogarea să reflecte cuvântul cheie pe care îl specificaţi în linia de comandă când rulaţi scriptul.

Înainte de a putea rula scriptul, trebuie să-l faceţi executabil:

```
% chmod +x interese.sh
```

Acum, nu mai trebuie să editaţi scriptul de fiecare dată când îl rulaţi. Pur şi simplu indicaţi direct în linia de comandă ce anume căutaţi:

```
% interese.sh criza
% interese.sh Jefferson
```

Crearea de noi înregistrări folosind date existente

Se pot adăuga noi înregistrări într-un tabel, rând cu rând, folosind instrucţiunea INSERT dar, după crearea câtorva înregistrări prin tastarea manuală a instrucţiunilor INSERT, majoritatea dintre noi suntem convinşi că trebuie să existe şi o metodă mai bună de a executa această operaţie. O alternativă este utilizarea unui fişier care conţine numai valori de date şi apoi încărcarea de înregistrări din acel fişier folosind instrucţiunea LOAD DATA sau utilitarul mysqlimport.

Deseori, creaţi fişierul de date folosind date care deja există într-un alt format. Informaţiile dumneavoastră pot fi conţinute într-un program de calcul tabelar sau poate că se află într-o altă bază de date şi doriţi să le transferaţi către MySQL. Pentru a nu complica această discuţie, voi presupune că datele dumneavoastră se află într-o foaie de calcul tabelar din calculatorul dumneavoastră de birou.

Pentru a transfera datele din foaia de calcul din calculatorul dumneavoastră într-un fişier din contul dumneavoastră UNIX, puteţi folosi tehnica de copiere şi lipire, alături de Telnet. Iată cum procedaţi:

1. Deschideţi o conexiune Telnet spre contul dumneavoastră UNIX. Sub Mac OS, puteţi folosi o aplicaţie precum BetterTelnet sau NCSA Telnet. Sub Windows, puteţi folosi programul Telnet standard.
2. Deschideţi foaia de calcul tabelar, selectaţi blocul de date pe care doriţi să-l transferaţi şi copiaţi-l.
3. În fereastra Telnet, tastaţi următoarea comandă pentru a începe capturarea datelor în fişierul data.txt:

   ```
   % cat > data.txt
   ```

   Comanda cat aşteaptă datele de intrare.
4. Lipiţi datele pe care le-aţi copiat din foaia de calcul în fereastra Telnet. Programul cat crede că introduceţi personal informaţiile şi le scrie îndatoritor în fişierul data.txt.
5. După ce toate datele lipite au fost scrise în fişier, apăsaţi pe tasta Enter în cazul în care cursorul ajunge la sfârşitul unei linii de date, nu la începutul unei linii noi. Apoi, apăsaţi pe combinaţia de taste Ctrl-D pentru a indica „sfârşitul fişierului". Programul cat încetează să mai aştepte date de intrare şi închide fişierul.

În acest moment, aveţi un fişier data.txt care conţine blocul de date pe care l-aţi selectat în foaia dumneavoastră de calcul tabelar şi care este gata de a fi încărcat în baza dumneavoastră de date cu LOAD DATA sau mysqlimport.

Copierea şi lipirea reprezintă o metodă rapidă şi simplă de a transfera date într-un fişier UNIX, dar este mai adecvată pentru seturi mai mici de date. Cantităţile mai mari de date pot depăşi limitele bufferului de copiere al sistemului dumneavoastră. În asemenea cazuri, se recomandă salvarea foii de calcul tabelar în format text simplu (delimitat prin tabulatori). Apoi, puteţi transfera fişierul din calculatorul dumneavoastră în contul UNIX folosind FTP. Transferaţi fişierul în mod text (nu în mod binar sau imagine), astfel încât caracterele sfârşit de rând să fie convertite în caractere UNIX de sfârşit de rând. (UNIX foloseşte caractere salt la linie nouă, Mac OS foloseşte caractere retur de car, iar Windows foloseşte perechea de caractere retur de car-salt la linie nouă.) Puteţi indica instrucţiunii LOAD DATA sau programului mysqlimport tipul caracterului sfârşit de rând pe care să-l aştepte, dar în UNIX este mai uşor de lucrat cu fişierul dacă acesta conţine caractere de salt la linie nouă.

După transferul fişierului, este bine să verificaţi dacă are sau nu rânduri vide la sfârşit. Dacă are, trebuie să le ştergeţi, deoarece altfel se vor transforma în înregistrări vide sau deformate atunci când încărcaţi fişierul în baza de date.

În fişierele salvate în format text simplu dintr-o foaie de calcul tabelar, valorile care conţin spaţii pot fi delimitate prin ghilimele. Pentru a elimina ghilimelele atunci când încărcaţi fişierul în baza dumneavoastră de date, folosiţi clauza `FIELDS ENCLOSED BY` pentru instrucţiunea `LOAD DATA`, respectiv opţiunea `--fields-enclosed-by` pentru `mysqlimport`. Vezi informaţiile despre instrucţiunea `LOAD DATA` din Anexa D pentru a obţine mai multe detalii.

De aici, încotro?

Acum ştiţi destule despre MySQL. Puteţi configura o bază de date şi crea tabele. Puteţi insera înregistrări în aceste tabele, le puteţi regăsi în diverse moduri, le puteţi modifica sau şterge. Dar mai aveţi încă multe de învăţat despre MySQL; manualul din acest capitol a reprezentat numai o mică introducere. Puteţi vedea acest lucru examinând starea bazei noastre de date exemplu. Am creat baza de date şi tabelele sale şi am populat-o cu unele date iniţiale. Pe parcursul acestui proces am văzut modul de scriere al unora dintre interogările de care avem nevoie pentru a răspunde la întrebări privind informaţiile din baza de date. Mai sunt, însă, multe de făcut...

De exemplu, nu dispunem de nici o modalitate interactivă convenabilă de a introduce noi înregistrări de punctaje în cadrul proiectului de evidenţă a rezultatelor şcolare, respectiv înregistrări cu noii membri în catalogul Ligii istorice. Nu avem nici o metodă convenabilă de a edita înregistrările existente. Şi tot nu putem genera formularele tipărite sau în format electronic pentru catalogul Ligii. Aceste sarcini, precum şi altele, vor fi abordate din nou în capitolele următoare, mai ales în capitolele 7, „Interfaţa API pentru Perl DBI", respectiv 8, „Interfaţa API pentru PHP".

Drumul pe care veţi merge în continuare printre paginile acestei cărţi depinde de domeniul dumneavoastră de interes. Dacă doriţi să vedeţi modul în care am finalizat proiectele legate de Liga istorică şi de evidenţa rezultatelor şcolare, Partea a II-a se referă la programarea în MySQL. Dacă veţi îndeplini funcţia de administrator MySQL pentru situl dumneavoastră, Partea a III-a a volumului de faţă tratează despre sarcinile administrative. Totuşi, eu vă recomand să acumulaţi mai întâi cunoştinţe suplimentare de ordin general despre utilizarea sistemului MySQL, citind celelalte capitole din Partea I. Aceste capitole discută despre modul de manipulare a datelor în MySQL, furnizează informaţii suplimentare despre sintaxă şi despre utilizarea instrucţiunilor de interogare şi vă prezintă modalităţi de rulare mai rapidă a interogărilor. O bază solidă de cunoştinţe legate de aceste subiecte vă va fi de folos indiferent de contextul în care folosiţi MySQL: rulaţi programul `mysql`, vă scrieţi propriile programe sau îndepliniţi atribuţiile unui administrator de baze de date.

CAPITOLUL 2

Lucrul cu date în MySQL şi SQL

Practic, tot ceea ce faceţi în MySQL implică, într-un fel sau altul, date, deoarece rostul unui sistem de gestiune a bazelor de date este, prin definiţie, acela de gestiune a datelor. Chiar şi o simplă instrucţiune `SELECT 1` implică o evaluare a unei expresii, cu scopul de a produce o valoare a unei date întregi.

Fiecare valoare a unei date din MySQL are un tip. De exemplu, `37.4` este un număr, iar `"abc"` este un şir. Uneori, tipurile de date sunt explicite, ca atunci când emiteţi o instrucţiune `CREATE TABLE` care specifică tipul fiecărei coloane pe care o declaraţi ca parte a tabelului:

```
CREATE TABLE tabelul_meu
(
    int_col INT,          /* coloana cu valori intregi */
    sir_col CHAR(20),     /* coloana cu valori sir */
    data_col DATE         /* coloana cu valori date */
)
```

În alte situaţii, tipurile de date sunt implicite, cum este atunci când faceţi referire la valori literale dintr-o expresie, când transferaţi valori unei funcţii sau când folosiţi valoarea returnată din acea funcţie:

```
INSERT INTO tabelul_meu (int_col,sir_col,data_col)
VALUES(14,CONCAT("a","b"),19990115)
```

Această instrucţiune `INSERT` efectuează următoarele operaţii, din care toate implică tipuri de date:

- Atribuie valoarea `14` coloanei întregi[1] `int_col`.
- Transferă valorile şir `"a"` şi `"b"` funcţiei `CONCAT()`. Această funcţie returnează valoarea şir `"ab"`, care este repartizată în coloana şir `sir_col`.
- Atribuie valoarea întreagă `19990115` coloanei de tip dată `data_col`. Atribuirea implică o neconcordanţă între tipuri, deci MySQL converteşte şirul `19990115` în data `"1999-01-15"`; acest fapt arată că MySQL execută conversia automată a tipului.

Pentru a folosi MySQL în mod eficient, este esenţial să înţelegeţi modul de manipulare a datelor în MySQL. Acest capitol descrie tipurile de date pe care le poate manipula MySQL şi discută problemele implicate de lucrul cu aceste tipuri:

---

[1] Prin *coloană întreagă* se va înţelege *coloană care conţine valori numere întregi*. S-a preferat în traducere prima formă, din motive de concizie. La fel de va proceda cu expresii gen *coloană şir* şi altele. – N.T.

- Categoriile generale de valori pe care le poate reprezenta MySQL, inclusiv valoarea NULL.
- Tipurile concrete de date pe care le furnizează MySQL pentru coloanele tabelelor, precum și proprietățile care caracterizează fiecare tip de coloană. Unele dintre tipurile de coloane MySQL au un caracter destul de general, cum este tipul șir CHAR. Alte tipuri, cum este TIMESTAMP, au comportări speciale, pe care trebuie să le înțelegeți pentru a evita surprizele.
- Alegerea adecvată a tipului de coloană pentru tabelele dumneavoastră. Este important să știți modul de a alege tipul cel mai adecvat intereselor dumneavoastră când construiți un tabel, precum și să alegeți între un tip și altul atunci când se pot aplica mai multe tipuri pentru categoria de valori pe care doriți să o stocați.
- Regulile MySQL pentru evaluarea expresiilor. MySQL oferă o gamă largă de operatori și funcții pe care le puteți folosi pentru a regăsi, afișa și manipula date. Regulile pentru evaluarea expresiilor includ regulile care controlează conversiile de tip, atunci când o valoare de un tip este folosită într-un context care impune o valoare de un alt tip.

Este important să înțelegeți când se produce și cum funcționează conversia de tip; unele conversii nu au nici un sens și generează valori fără nici o semnificație. Prin repartizarea șirului "13" într-o coloană întreagă se obține valoarea 13, dar repartizarea șirului "abc" în coloana respectivă are ca rezultat valoarea 0, deoarece "abc" nu arată ca un număr. Mai rău, dacă efectuați o comparație fără a cunoaște regulile de conversie puteți provoca pagube serioase, cum ar fi actualizarea sau ștergerea tuturor rândurilor dintr-un tabel atunci când intenționați să modificați numai câteva rânduri.

Două anexe conțin informații suplimentare despre tipurile de coloane, operatorii și funcțiile MySQL. Acestea sunt Anexa B, „Referință de tipuri de coloane“, respectiv Anexa C, „Referință de operatori și funcții“.

Tipuri de date MySQL

MySQL „cunoaște“ numeroase tipuri de date, în speță categorii generale în care pot fi reprezentate diverse valori.

Valori numerice

Numerele sunt valori precum 48 sau 193.62. MySQL înțelege numere specificate ca întregi (fără parte fracționară) sau valori cu virgulă mobilă (cu parte fracționară). Întregii pot fi specificați în format zecimal sau hexazecimal.

Un întreg constă dintr-o secvență de cifre. Un întreg specificat în formă hexazecimală constă din secvența 0x urmată de una sau mai multe cifre hexazecimale (cuprinse între 0-9, respectiv a-f). De exemplu, 0x0a este 10 în format zecimal, iar 0xffff este 65535 în format zecimal. Cifrele hexazecimale non-numerice pot fi specificate cu majuscule sau minuscule, dar secvența de început 0x nu poate fi scrisă sub forma 0X. Cu alte cuvinte, 0x0a și 0x0A sunt corecte, dar 0X0a și 0X0A nu sunt.

Un număr cu virgulă mobilă constă dintr-o secvență de cifre, o virgulă de separare a părții fracționare și o altă secvență de cifre. Una dintre cele două secvențe de cifre poate fi vidă, dar nu ambele simultan.

MySQL înțelege notația științifică. Aceasta este indicată prin inserția imediat după un număr întreg sau în virgulă mobilă a literei e sau E, unui caracter semn (+ sau –) și unui exponent întreg. 1.34E+12 și 43.27e-1 reprezintă numere scrise în notație științifică corectă. Pe de altă parte, 1.34E12 *nu* este scris corect, deoarece lipsește caracterul semn înaintea exponentului. Numerele hexazecimale nu pot fi folosite în notațiile științifice; caracterul e care se află la începutul exponentului este, de asemenea, o cifră hexazecimală corectă și astfel se pot crea confuzii.

Numerele pot fi precedate de un semn minus (–) pentru a semnala o valoare negativă.

Valori șir (caracter)

Șirurile sunt valori ca "Bucuresti, Romania" sau "pacientul se simte mai bine". Pentru a delimita o valoare șir se pot folosi ghilimele simple sau duble.

În interiorul șirurilor se pot folosi numeroase secvențe escape, care se pot folosi pentru a preciza caractere speciale, așa cum se poate vedea în tabelul 2.1. Fiecare secvență începe cu un caracter backslash (\) pentru a semnifica o excepție temporară de la regulile obișnuite de interpretare a caracterelor. Rețineți că un octet NUL nu este unul și același lucru cu o valoare NULL; NUL este un octet de valoare zero, în timp ce NULL semnifică absența unei valori.

Tabelul 2.1 Secvențe escape din șiruri

Secvență	Semnificație
\0	NUL (ASCII 0)
\'	Ghilimele simple
\"	Ghilimele duble
\b	Ștergere caracter anterior
\n	Caracter linie nouă
\r	Retur de car
\t	Caracter de tabulare
\\	Backslash

Pentru a include ghilimele într-un șir, puteți proceda astfel:

- Dublați ghilimelele, dacă șirul este inclus între ghilimele folosind același caracter:

```
'Anii ''70'
"El zise: ""Ti-am spus eu!"""
```

- Încadrați șirul folosind celelalte ghilimele; în acest caz, nu dublați ghilimelele din șir:

```
"Anii '70"
'El zise: "Ti-am spus eu!"'
```

- Anulați semnificația ghilimelelor cu un caracter backslash; această metodă este aplicabilă indiferent de tipul ghilimelelor folosite pentru a încadra șirul:

```
'Anii \'70'
'Anii \'70"
```

```
"El zise: \"Ti-am spus eu!\""
'El zise: \"Ti-am spus eu!\"'
```

În contextele șirurilor, se pot folosi constantele hexazecimale pentru a specifica valorile de tip șir. Sintaxa este cea descrisă anterior pentru valorile numerice, dar perechile de cifre hexazecimale sunt interpretate drept coduri ASCII și sunt convertite în caractere. Rezultatul este folosit sub formă de șir. De exemplu, când este interpretat sub formă de șir, `0x616263` este `"abc"`.

Valori de tip dată și oră (temporale)

Datele și orele sunt valori precum `"1999-06-17"` sau `"12:30:43"`. De asemenea, MySQL înțelege valorile dată/oră combinate, precum `"1999-06-17 12:30:43"`. Rețineți faptul că MySQL reprezintă datele în ordinea an-lună-zi. Acest lucru îi surprinde deseori pe începătorii în materie de MySQL, deși acest format este standardul SQL ANSI. Puteți afișa valorile datelor în orice mod doriți folosind funcția `DATE_FORMAT()`, dar formatul de afișare prestabilit afișează mai întâi anul, iar valorile de intrare *trebuie* specificate începând cu anul.

Valoarea NULL

`NULL` este un fel de valoare „fără tip". În general, este folosită cu semnificația „fără valoare", „valoare necunoscută", „valoare lipsă", „în afara domeniului", „nici una din precedentele opțiuni" și așa mai departe. Puteți insera valori `NULL` în tabele, le puteți regăsi din tabele și puteți testa dacă o valoare este sau nu `NULL`. Nu puteți efectua operații aritmetice cu valori `NULL`. (Dacă încercați, rezultatul este `NULL`.)

Tipuri de coloane MySQL

Fiecare tabel dintr-o bază de date este alcătuit dintr-una sau mai multe coloane. Când creați un tabel folosind o instrucțiune `CREATE TABLE`, specificați un tip pentru fiecare coloană. Un tip de coloană este mai specific decât un tip de date, care este numai o categorie generală de genul „număr" sau „șir". Un tip de coloană caracterizează cu exactitate categoria de valori pe care o poate conține o coloană a unui tabel dat, cum este `SMALLINT` sau `VARCHAR(32)`.

Tipurile de coloane MySQL reprezintă mijloacele prin care descrieți categoriile de valori pe care le conțin coloanele unui tabel, care, la rândul lor, determină modul în care MySQL tratează aceste valori. De exemplu, dacă aveți valori numerice, le puteți stoca folosind un tip de coloană numeric sau șir, dar MySQL va trata valorile oarecum diferit, în funcție de modul în care le stocați. Fiecare tip de coloană are numeroase caracteristici:

- Ce tip de valori puteți stoca în coloana respectivă
- Spațiul pe care îl ocupă valorile, precum și dacă valorile sunt de lungime fixă (toate valorile de acel tip ocupă aceeași cantitate de spațiu) sau de lungime variabilă (cantitatea de spațiu depinde de valoarea particulară stocată)
- Modul de comparare și stocare a valorilor din tipul respectiv
- Dacă tipul respectiv acceptă sau nu valori `NULL`
- Dacă tipul respectiv poate fi indexat sau nu

Vom examina succint tipurile de coloană MySQL pentru a avea o imagine de ansamblu, după care vom discuta mai detaliat proprietățile care caracterizează fiecare tip.

O examinare a tipurilor de coloane

MySQL furnizează tipuri de coloane pentru valori din toate categoriile generale de tipuri de date, cu excepția valorii `NULL`. Valoarea `NULL` acoperă toate tipurile, în sensul că posibilitatea unei coloane de a conține sau nu valori `NULL` este tratată ca atribut de tip.

MySQL are tipuri de coloane atât pentru valorile întregi, cât și pentru cele cu virgulă mobilă, așa cum se poate vedea în tabelul 2.2. Coloanele întregi pot fi cu sau fără semn. Un atribut special permite valorilor din coloanele întregi să fie generate automat, ceea ce este util în aplicații care impun numere dintr-o secvență sau de identificare unice.

Tabelul 2.2 Tipuri de coloane numerice

Numele tipului	Semnificație
`TINYINT`	Un întreg foarte mic
`SMALLINT`	Un întreg mic
`MEDIUMINT`	Un întreg de dimensiune medie
`INT`	Un întreg standard
`BIGINT`	Un întreg mare
`FLOAT`	Un număr cu virgulă mobilă în precizie simplă
`DOUBLE`	Un număr cu virgulă mobilă în dublă precizie
`DECIMAL`	Un număr cu virgulă mobilă, reprezentat sub formă de șir

Tipurile de coloane șir din MySQL sunt prezentate în tabelul 2.3. Șirurile pot conține orice, chiar și date binare arbitrare, cum sunt imaginile sau sunetele. Șirurile pot fi comparate în funcție de sensibilitatea lor la diferența între majuscule și minuscule. De asemenea, puteți efectua cu șiruri stabilirea de corespondențe cu un model. (De fapt, în MySQL puteți efectua stabilirea de corespondențe cu un model pentru orice tip de coloane, dar această operație se execută cel mai frecvent cu tipuri șir.)

Tabelul 2.3 Tipuri de coloane șir

Numele tipului	Semnificație
`CHAR`	Un șir de caractere de lungime fixă
`VARCHAR`	Un șir de caractere de lungime variabilă
`TINYBLOB`	Un BLOB (obiect binar mare) foarte mic
`BLOB`	Un BLOB mic
`MEDIUMBLOB`	Un BLOB de dimensiuni medii

Continuare

Tabelul 2.3 Continuare

Numele tipului	Semnificație
LONGBLOB	Un BLOB de dimensiuni mari
TINYTEXT	Un șir text foarte mic
TEXT	Un șir text mic
MEDIUMTEXT	Un șir text de dimensiuni medii
LONGTEXT	Un șir text de mari dimensiuni
ENUM	O enumerare; coloanelor li se poate atribui un membru al enumerării
SET	Un set; coloanelor li se pot atribui mai mulți membri ai unui set

Tipurile date și oră din MySQL sunt prezentate în tabelul 2.4. Pentru valori temporale, MySQL furnizează tipuri pentru date (cu sau fără oră), ore și amprente de timp (un tip special, care vă permite să detectați data și ora efectuării ultimei modificări într-o înregistrare). De asemenea, există un tip pentru reprezentarea eficientă a valorilor anilor, atunci când nu aveți nevoie de o dată completă.

Tabelul 2.4 Tipuri de coloane dată și oră

Numele tipului	Semnificație
DATE	O valoare pentru dată, în format "*AAAA-LL-ZZ*"
TIME	O valoare pentru oră, în format "*hh:mm:ss*"
DATETIME	O valoare pentru dată și oră, în format "*AAAA-LL-ZZ hh:mm:ss*"
TIMESTAMP	O valoare pentru amprenta de timp, în format *AAAALLZZhhmmss*
YEAR	O valoare pentru an, în format AAAA

Pentru a crea un tabel, emiteți o instrucțiune `CREATE TABLE` și specificați o listă a coloanelor care alcătuiesc tabelul. Fiecare coloană are un nume și un tip, iar la fiecare tip se pot asocia mai multe atribute. Iată un exemplu care creează un tabel denumit `tabelul_meu`, cu trei coloane, denumite `f`, `c` și `i`:

```
CREATE TABLE tabelul_meu
(
    f FLOAT(10,4),
    c CHAR(15) NOT NULL DEFAULT "nimic",
    i TINYINT UNSIGNED NULL
)
```

Sintaxa pentru declararea unei coloane este următoarea:

```
nume_col tip_col [atribute_coloana] [atribute_generale]
```

Numele coloanei este dat de `nume_col`. Numele coloanelor pot avea o lungime de maximum 64 de caractere și pot fi alcătuite din caractere alfanumerice, precum și din caracterul de subliniere și simbolul dolarului (în speță `_` și `$`). Un nume de coloană poate începe cu orice caracter admis într-un nume, inclusiv cu o cifră. Totuși, un nume nu poate fi compus integral din cifre, deoarece astfel ar deveni practic confundabil cu un număr. Cuvinte precum `SELECT`, `DELETE` sau `CREATE` sunt rezervate și nu pot fi folosite drept nume de coloană. Totuși, numele funcțiilor (cuvinte precum `POS` și `MIN`) nu sunt rezervate și pot fi folosite.

Tipul de coloană `tip_col` precizează categoria exactă de valori pe care le poate conține coloana. Specificatorul de tip mai poate indica și lungimea maximă a valorilor pe care le stocați în coloană. Pentru unele tipuri, specificați lungimea în mod explicit sub formă numerică. Pentru altele, lungimea se deduce din numele tipului. De exemplu, `CHAR(10)` specifică o lungime explicită de 10 caractere, în timp ce valorile `TINYBLOB` au o lungime maximă implicită de 255 caractere. Unii dintre specificatorii de tip vă permit să indicați o lățime maximă afișată (numărul de caractere de utilizat pentru afișarea valorilor). Tipurile de coloane cu virgulă mobilă permit specificarea numărului de cifre după virgulă, astfel încât să puteți controla nivelul de precizie al valorilor.

După tipul coloanei, puteți specifica atribute opționale specifice unui anumit tip, precum și atribute de ordin mai general. Atributele funcționează ca modificatori de tip. Acestea determină programul MySQL să modifice într-un fel sau altul tratamentul aplicat valorilor din coloană:

- Atributele permise specifice tipului depind de tipul de coloană pe care îl alegeți. De exemplu, `UNSIGNED` este permis numai pentru tipurile întregi, iar `BINARY` este permis numai pentru tipurile `CHAR` și `VARCHAR`.
- Atributele generale pot fi date pentru orice tip de coloană, cu anumite excepții. Puteți specifica `NULL` sau `NOT NULL` pentru a preciza dacă o coloană poate conține sau nu valori `NULL`. De asemenea, puteți specifica `DEFAULT valoare_prestabilita` pentru a arăta că o coloană trebuie să primească valoarea `valoare_prestabilita` atunci când este creat un rând nou fără a se specifica în mod explicit valoarea coloanei. Valoarea `valoare_prestabilita` trebuie să fie o constantă; nu poate fi o expresie și nici nu poate face referire la alte coloane. Nu puteți specifica o valoare prestabilită pentru coloanele `BLOB` sau `TEXT`.

Dacă sunt precizate mai multe atribute specifice unei anumite coloane, acestea pot fi precizate în orice ordine, cu condiția ca acestea să respecte tipul coloanei și să preceadă toate atributele generale. Similar, dacă sunt date mai multe atribute generale, ele pot fi specificate în orice ordine, cu condiția să respecte tipul coloanei și să fie plasate după toate atributele specifice de coloană existente.

În continuare, această secțiune discută despre fiecare dintre tipurile de coloană MySQL, pentru a prezenta sintaxa utilizată la declararea tipului și proprietățile care o caracterizează, cum ar fi cerințele privind domeniul și capacitatea de stocare. Specificațiile de tip sunt prezentate așa cum sunt utilizate acestea în instrucțiunile `CREATE TABLE`. Informațiile opționale sunt încadrate între paranteze drepte (`[]`). De exemplu, sintaxa `MEDIUMINT[(M)]` arată că lățimea maximă de afișare, specificată sub forma `(M)`, este opțională. Pe de altă parte, pentru `CHAR(M)`, lipsa parantezelor arată că `(M)` este obligatorie.

Tipuri de coloane numerice

Tipurile de coloane numerice din MySQL se încadrează în două categorii generale:

- **Tipuri întregi.** Pentru numerele fără parte fracţionară, precum `1`, `43`, `-3`, `0` sau `-798432`. Puteţi folosi coloanele întregi pentru date reprezentate de numere întregi, precum greutatea aproximată la cea mai apropiată cantitate în kilograme, înălţimea aproximată la valoarea în centimetri cea mai apropiată, numărul de stele dintr-o galaxie, numărul de persoane dintr-o gospodărie sau numărul de bacterii dintr-un vas de cultură.
- **Tipuri cu virgulă mobilă.** Pentru numere care pot avea o parte fracţionară, precum `3.14159`, `-.00273`, `-4.78` sau `39.3E+4`. Puteţi folosi tipuri de coloane cu virgulă mobilă pentru valori care pot avea o parte fracţionară sau care sunt extrem de mari sau de mici. Printre tipurile de date care pot fi reprezentate sub formă de valori cu virgulă mobilă sunt: producţia agricolă medie, distanţele, valorile monetare (preţul unui articol sau un salariu), rata şomajului sau preţurile acţiunilor la bursă. Acestea sunt tratate sub formă de valori cu virgulă mobilă cu o parte fracţionară zero.

Numele şi domeniul fiecărui tip numeric sunt prezentate în tabelul 2.5. Cantitatea de spaţiu necesară pentru valorile de fiecare tip este prezentată în tabelul 2.6.

Instrucţiunea CREATE TABLE

Exemplele folosite în acest capitol utilizează *in extenso* instrucţiunea `CREATE TABLE`. Instrucţiunea ar trebui să vă fie destul de cunoscută, deoarece am folosit-o în secţiunea de manual a capitolului 1, „Introducere în MySQL şi SQL". De asemenea, vezi informaţiile despre instrucţiunea `CREATE TABLE` din Anexa D, „Referinţă de sintaxă SQL".

Tabelul 2.5 Domenii de existenţă ale tipurilor de coloane numerice

Specificaţie de tip	Domeniu
`TINYINT[(M)]`	Valori cu semn: între -128 şi 127 (între -2^7 şi 2^7-1) Valori fără semn între 0 şi 255 (între 0 şi 2^8-1)
`SMALLINT[(M)]`	Valori cu semn: între -32768 şi 32767 (între -2^{15} şi $2^{15}-1$) Valori fără semn: între 0 şi 65535 (între 0 şi $2^{16}-1$)
`MEDIUMINT[(M)]`	Valori cu semn: între -8388608 şi 8388607 (între -2^{23} şi $2^{23}-1$) Valori fără semn: între 0 şi 16777215 (între 0 şi $2^{24}-1$)
`INT[(M)]`	Valori cu semn: între - 2147483648 şi 2147483647 (între -2^{31} şi $2^{31}-1$) Valori fără semn: între 0 şi 4294967295 (între 0 şi $2^{32}-1$)
`BIGINT[(M)]`	Valori cu semn: între -9223372036854775808 şi 9223372036854775807 (între -2^{63} şi $2^{63}-1$) Valori fără semn: între 0 şi 18446744073709551615 (între 0 şi $2^{64}-1$)
`FLOAT[(M,D)]`, `FLOAT(4)`	Valori minime diferite de zero: ±1.175494351E-38 Valori maxime diferite de zero: ±3.402823466E+38
`DOUBLE[(M,D)]`, `FLOAT(8)`	Valori minime diferite de zero: ±2.2250738585072014E-308 Valori maxime diferite de zero: ±1.7976931348623157E+308
`DECIMAL(M,D)`	Variază; domeniul depinde de *M* şi *D*.

Tabelul 2.6 Necesităţi de spaţiu de stocare pentru tipurile de coloane numerice

Specificaţie de tip	Spaţiu necesar
`TINYINT[(M)]`	1 octet
`SMALLINT[(M)]`	2 octeţi
`MEDIUMINT[(M)]`	3 octeţi
`INT[(M)]`	4 octeţi
`BIGINT[(M)]`	8 octeţi
`FLOAT[(M,D)]`, `FLOAT(4)`	4 octeţi
`DOUBLE[(M,D)]`, `FLOAT(8)`	8 octeţi
`DECIMAL(M,D)`	*M* octeţi (versiune MySQL anterioară versiunii 3.23), *M*+2 octeţi (versiune MySQL ulterioară versiunii 3.23 sau chiar versiunea 3.23)

MySQL furnizează cinci tipuri întregi: `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT` şi `BIGINT`. `INTEGER` este un sinonim pentru `INT`. Aceste tipuri variază din punctul de vedere al domeniului de valori pe care îl pot reprezenta. Coloanele întregi pot fi declarate ca `UNSIGNED` pentru a interzice valorile negative; astfel, domeniul pentru coloană este deplasat în sus pe axa numerelor, începând astfel de la `0`. De asemenea, tipurile variază din punct de vedere al cantităţii de spaţiu necesare. Tipurile cu un domeniu mai extins necesită un spaţiu de stocare mai mare.

MySQL furnizează trei tipuri cu virgulă mobilă: `FLOAT`, `DOUBLE` şi `DECIMAL`. Spre deosebire de tipurile întregi, tipurile cu virgulă mobilă nu pot fi `UNSIGNED`, iar domeniul lor este diferit de acela al tipurilor întregi, prin aceea că tipul respectiv nu poate reprezenta numai o valoare maximă, ci şi o valoare minimă diferită de zero. Valorile minime furnizează o măsură a preciziei tipului, ceea ce este deseori important pentru înregistrarea datelor ştiinţifice. (Există, desigur, valori maxime şi minime negative corespunzătoare.)

`DOUBLE PRECISION[(M,D)]` şi `REAL[(M,D)]` sunt sinonime pentru `DOUBLE[(M,D)]`. `NUMERIC(M,D)`este sinonim cu `DECIMAL(M,D)`. `FLOAT(4)` şi `FLOAT(8)` sunt furnizate pentru compatibilitatea cu ODBC. Înainte de MySQL 3.23, acestea erau sinonime cu `FLOAT(10,2)`, respectiv `DOUBLE(16,4)`. Începând de la MySQL 3.23, `FLOAT(4)` şi `FLOAT(8)` îşi au propriul comportament, descris pe scurt în continuare.

Când alegeţi un tip numeric, vă gândiţi la domeniul de valori pe care trebuie să-l reprezentaţi şi alegeţi cel mai mic tip care va acoperi domeniul. Prin alegerea unui tip mai mare se pierde spaţiu, rezultând astfel tabele inutil de mari, care nu pot fi prelucrate la fel de eficient ca şi atunci când aţi fi ales un tip mai mic. Pentru valori întregi, `TINYINT` este optim dacă domeniul de valori al datelor dumneavoastră este redus, cum ar fi vârsta unei persoane sau numărul de fraţi sau surori. `MEDIUMINT` poate reprezenta milioane de valori şi poate fi folosit pentru mult mai multe tipuri de valori, cu preţul unui spaţiu de stocare ceva mai mare. `BIGINT` are cel mai mare domeniu dintre toate tipurile, dar necesită un spaţiu de stocare dublu faţă de cel mai mic tip întreg următor (`INT`) şi trebuie să fie folosit numai atunci când este într-adevăr nevoie. Pentru valori cu virgulă mobilă, `DOUBLE` ocupă de două ori mai mult spaţiu decât `FLOAT`. Cu excepţia situaţiilor când aveţi nevoie de o precizie excepţională sau de un domeniu de valori extrem de mare, probabil că vă puteţi reprezenta datele la numai jumătate din spaţiul de stocare, folosind `FLOAT`.

Când declarați o coloană de tip întreg, puteți specifica o dimensiune opțională de afișare `M`. Dacă este dat, `M` trebuie să fie un întreg cuprins între 1 și 255 și reprezintă numărul de caractere folosit pentru a afișa valorile din coloană. De exemplu, `MEDIUMINT(4)` specifică o coloană `MEDIUMINT` cu o lățime de afișare egală cu 4. Dacă declarați o coloană de tip întreg fără o lățime explicită, este atribuită o lățime prestabilită. Valorile prestabilite constituie lungimile valorilor „celor mai lungi" pentru fiecare tip. Dacă reprezentarea care se poate afișa a unei anumite valori necesită mai mult de `M` caractere, atunci se va afișa valoarea totală; valorile nu vor fi „amputate" pentru a se încadra în limita celor `M` caractere.

Pentru fiecare tip de coloană cu virgulă mobilă, puteți specifica o dimensiune maximă de afișare `M` și numărul de cifre după virgulă `D`. Valoarea lui `M` trebuie să fie cuprinsă între 1 și 255. Valoarea lui `D` poate fi cuprinsă între 0 și 30, dar nu trebuie să depășească `M-2`. (Dacă sunteți un cunoscător al terminologiei ODBC, `M` și `D` corespund conceptelor ODBC de „precizie" și „scară".) `M` și `D` sunt opționale pentru `FLOAT` și `DOUBLE`, dar sunt obligatorii pentru `DECIMAL`.

Acolo unde `M` și `D` sunt opționale, în caz de omisiune sunt folosite valori prestabilite. Următoarea instrucțiune creează un tabel pentru a ilustra valorile prestabilite ale opțiunilor `M` și `D` pentru tipurile de coloane numerice (`DECIMAL` nu este inclus, deoarece `M` și `D` nu sunt opționale pentru tipul respectiv):

```
CREATE TABLE tabelul_meu
(
    itiny TINYINT, itiny_u TINYINT UNSIGNED,
    ismall SMALLINT, ismall_u SMALLINT UNSIGNED,
    imedium MEDIUMINT, imedium_u MEDIUMINT UNSIGNED,
    ireg INT, ireg_u INT UNSIGNED,
    ibig BIGINT, ibig_u BIGINT UNSIGNED,
    fp_single FLOAT, fp_double DOUBLE
)
```

Dacă emiteți o instrucțiune `DESCRIBE tabelul_meu` după crearea tabelului, coloanele `Field` și `Type` ale datelor de ieșire se vor prezenta astfel[2]:

Field	Type
itiny	tinyint(4)
itiny_u	tinyint(3) unsigned
ismall	smallint(6)
ismall_u	smallint(5) unsigned
imedium	mediumint(9)
imedium_u	mediumint(8) unsigned
ireg	int(11)
ireg_u	int(10) unsigned
ibig	bigint(20)
ibig_u	bigint(20) unsigned
fp_single	float(10,2)
fp_double	float(16,4)

[2] Lățimea de afișare pentru `BIGINT` va fi 21 (nu 20) dacă rulați această interogare folosind o versiune MySQL anterioară versiunii 3.23, datorită unei erori minore. – N.A.

Fiecare coloană numerică are un domeniu de valori determinat de tipul coloanei. Dacă încercați să inserați o valoare care se află în afara domeniului coloanei, se produce o trunchiere: MySQL „taie" valoarea corespunzător punctului final adecvat al domeniului și folosește rezultatul. La regăsirea valorilor nu se produce nici o trunchiere.

Trunchierea valorilor se produce în conformitate cu domeniul tipului coloanei, nu cu lățimea de afișare. De exemplu, o coloană `SMALLINT(3)` are o lățime de afișare egală cu 3 și un domeniu cuprins între `-32768` și `32767`. Valoarea `12345` este mai „lată" decât lățimea de afișare, dar se află în interiorul domeniului coloanei, deci este inserată fără „tăiere" și regăsită sub forma `12345`. Valoarea `99999` se află în afara domeniului, deci este redusă la `32767` când este inserată. La regăsirile ulterioare, valoarea apare sub forma `32767`.

În general, valorile atribuite unei coloane cu virgulă mobilă sunt rotunjite la numărul de zecimale indicat la specificarea coloanei. Dacă stocați valoarea `1.23456`[3] într-o coloană `FLOAT(8,1)`, rezultatul este `1.2`. Dacă stocați aceeași valoare într-o coloană `FLOAT(8,4)`, rezultatul este `1.2346`. Aceasta înseamnă că trebuie să declarați coloanele cu virgulă mobilă cu un număr de zecimale suficient de mare pentru a vă oferi valori la precizia dorită. Dacă aveți nevoie de o precizie de ordinul miimilor, nu declarați un tip cu numai două cifre după virgulă.

Excepția de la acest mod de manipulare a valorilor cu virgulă mobilă constă în aceea că, în MySQL 3.23, comportarea tipurilor `FLOAT(4)` și `FLOAT(8)` s-a modificat. Aceste două tipuri sunt acum de tip precizie simplă (4 octeți), respectiv dublă (8 octeți), care sunt tipuri cu virgulă mobilă autentice, în sensul că valorile sunt stocate așa cum sunt date, în cadrul limitelor impuse de componentele dumneavoastră hardware.

Tipul `DECIMAL` este diferit de `FLOAT` și `DOUBLE`, în sensul că valorile `DECIMAL` sunt, de fapt, stocate sub formă de șiruri. Domeniul maxim posibil pentru `DECIMAL` este același ca pentru `DOUBLE`, dar domeniul efectiv este determinat de valorile parametrilor `M` și `D`. Dacă `M` variază și `D` rămâne fix, domeniul crește dacă `M` crește. Acest fapt este ilustrat de primele trei rânduri ale tabelului 2.7. Dacă `M` rămâne fix *și* `D` variază, domeniul devine mai mic dacă `D` crește, deși precizia crește. Acest fapt este ilustrat de ultimele trei rânduri ale tabelului 2.7.

Tabelul 2.7 Influența valorilor parametrilor *M* și *D* asupra domeniului tipului `DECIMAL(M,D)`

Specificație de tip	Domeniu (pentru MySQL < 3.23)	Domeniu (pentru MySQL 3.23)
DECIMAL(4,1)	între -9,9 și 99,9	între -999,9 și 9999,9
DECIMAL(5,1)	între -99,9 și 999,9	între -9999,9 și 99999,9
DECIMAL(6,1)	între -999,9 și 9999,9	între -99999,9 și 999999,9
DECIMAL(6,2)	între -99,99 și 999,99	între -9999,99 și 99999,99
DECIMAL(6,3)	între -9,999 și 99,999	între -999,999 și 9999,999

Domeniul unui tip `DECIMAL` dat depinde de versiunea dumneavoastră de MySQL. Pentru versiunile MySQL anterioare versiunii 3.23, coloanele de tip `DECIMAL(M,D)` sunt stocate folosind `M` octeți per valoare, iar caracterul semn (dacă este necesar) și virgula sunt incluse în cei `M` octeți. Astfel, pentru un tip `DECIMAL(5,2)`, domeniul este cuprins între -9,99 și 99,99, deoarece în acest interval se încadrează toate valorile de 5 caractere posibile.

[3] În limba engleză, virgula zecimală se reprezintă ca punct zecimal. MySQL folosește acest sistem. – N.T.

Începând de la MySQL 3.23, valorile DECIMAL sunt manipulate în conformitate cu specificația ANSI, care precizează că un tip DECIMAL(M,D) trebuie să poată reprezenta toate valorile cu M cifre și D cifre după virgulă. De exemplu, DECIMAL(5,2) trebuie să poată reprezenta valori cuprinse între -999,99 și 999,99. Caracterul semn și virgula trebuie și ele stocate, deci valorile DECIMAL începând de la versiunea MySQL 3.23 folosesc M+2 octeți. Pentru DECIMAL(5,2) sunt necesari 7 octeți pentru „cea mai lungă" valoare (-999,99). La extremitatea pozitivă a domeniului, octetul de semn nu este necesar pentru memorarea unui caracter semn, deci MySQL îl folosește pentru a extinde domeniul dincolo de cel cerut prin specificația ANSI. Pentru DECIMAL(5,2), valoarea maximă poate fi 9999,99, deoarece sunt disponibili 7 octeți.

Pe scurt, domeniul tipului DECIMAL(M,D) pentru versiunile de la MySQL 3.23 „în sus" este echivalent cu domeniul tipului DECIMAL(M+2,D) din versiunile anterioare.

În toate versiunile de MySQL, dacă D este egal cu 0 pentru o coloană DECIMAL, virgula nu este stocată. Efectul este o extindere a domeniului coloanei cu un ordin de mărime suplimentar, deoarece octetul folosit în mod normal pentru stocarea virgulei se poate utiliza pentru o altă cifră.

Atributele tipului de coloană numerică

Atributul ZEROFILL poate fi specificat pentru toate tipurile numerice. Acesta determină completarea cu zerouri inițiale a valorilor afișate pentru coloană, pentru a se obține lățimea de afișare. Puteți folosi ZEROFILL când doriți să vă asigurați că valorile din coloane sunt întotdeauna afișate folosind un număr de cifre dat. De fapt, este mai bine să spunem „un număr *minim* dat de cifre", deoarece valorile cu un număr de cifre mai mare decât lățimea de afișare sunt afișate complet, fără a fi „tăiate". Puteți vedea acest lucru emițând următoarele instrucțiuni:

```
CREATE TABLE tabelul_meu (completare_zerouri INT(5) ZEROFILL)
INSERT INTO tabelul_meu VALUES(1),(100),(10000),(1000000)
SELECT completare_zerouri FROM tabelul_meu
```

Datele de ieșire ale instrucțiunii SELECT sunt prezentate în continuare. Observați că valoarea finală, care are un număr de cifre mai mare decât lățimea de afișare a coloanei, este afișată complet:

completare_zerouri
00001
00100
10000
1000000

Alte două atribute pot fi specificate numai pentru tipurile de coloane întregi:

- AUTO_INCREMENT. Folosiți atributul AUTO_INCREMENT când doriți să generați identificatori unici sau valori în serie. În mod normal, valorile AUTO_INCREMENT încep de la 1 și cresc cu o unitate pentru fiecare rând. Când inserați NULL într-o coloană AUTO_INCREMENT, MySQL inserează o valoare cu o unitate mai mare decât valoarea maximă curentă din acea coloană. Într-un tabel puteți avea maximum o coloană AUTO_INCREMENT.

  Pentru orice coloană unde doriți să folosiți AUTO_INCREMENT, coloana trebuie să fie declarată NOT NULL și, de asemenea, ca PRIMARY KEY sau drept cheie UNIQUE. De exemplu, puteți declara o asemenea coloană în oricare din următoarele moduri:

```
CREATE TABLE ai (i INT AUTO_INCREMENT NOT NULL PRIMARY KEY)
CREATE TABLE ai (i INT AUTO_INCREMENT NOT NULL, PRIMARY KEY (i))
CREATE TABLE ai (i INT AUTO_INCREMENT NOT NULL, UNIQUE(i))
```

  Comportarea atributului AUTO_INCREMENT este discutată mai aprofundat în paragraful „Lucrul cu secvențe".

- UNSIGNED. Acest atribut interzice valorile negative. Dacă o coloană primește atributul UNSIGNED, dimensiunea domeniului tipului de date de bază nu se modifică, ci pur și simplu este deplasată înainte (spre dreapta) pe axa numerelor. Gândiți-vă la această specificație de tabel:

```
CREATE TABLE tabelul_meu
(
    itiny TINYINT,
    itiny_u TINYINT UNSIGNED
)
```

  itiny și itiny_u sunt ambele coloane de tip TINYINT, cu un domeniu de 256 de valori, dar domeniul lui itiny este cuprins între –128 și 127, în timp ce domeniul lui itiny_u este cuprins între 0 și 255.

  UNSIGNED este util pentru coloane în care intenționați să stocați date care nu iau valori negative, cum sunt valorile populațiilor sau cifrele privind prezența la un curs. Dacă folosiți o coloană obișnuită, cu semn, pentru asemenea valori, folosiți numai jumătate din domeniul tipului de coloană. Când coloana primește atributul UNSIGNED, efectiv își dublează domeniul. Dacă folosiți coloana pentru numere în secvență, veți avea nevoie de un număr dublu de valori pentru a epuiza domeniul dacă declarați coloana cu atributul UNSIGNED.

După atributele pe care tocmai le-am descris, care sunt caracteristice coloanelor numerice, mai puteți specifica și atributele generale NULL și NOT NULL. Dacă nu specificați NULL sau NOT NULL, valoarea prestabilită este NULL. De asemenea, puteți specifica o valoare prestabilită folosind atributul DEFAULT. Dacă nu specificați o valoare prestabilită, aceasta va fi aleasă automat. Pentru toate tipurile de coloane numerice, valoarea prestabilită este NULL pentru coloanele care pot conține NULL, respectiv 0 pentru celelalte.

Exemplul următor creează un tabel cu trei coloane INT, cu valorile prestabilite -1, 1 și NULL:

```
CREATE TABLE t
(
    i1 INT DEFAULT -1,
    i2 INT DEFAULT 1,
    i3 INT DEFAULT NULL
)
```

Lucrul cu secvențe

Multe aplicații trebuie să folosească numere unice pentru motive legate de identificare. Necesitatea unicității valorilor survine într-un număr de contexte: numere de membri, numerotarea mostrelor sau a loturilor, identificatori de clienți, etichete ale rapoartelor privind hibele sau ale tichetelor de semnalare a problemelor și altele.

Mecanismul sistemului MySQL pentru furnizarea de numere unice folosește coloanele AUTO_INCREMENT. Acestea vă permit să generați automat numere în secvență. Din păcate, AUTO_INCREMENT este o facilitate uneori prost înțeleasă, fenomen determinat poate de modificările aduse acestei caracteristici în MySQL 3.23. Această secțiune descrie modul de comportare a coloanelor AUTO_INCREMENT astfel încât dumneavoastră să le puteți folosi în mod eficient, fără a cădea în capcanele care deseori iau oamenii prin surprindere. De asemenea, secțiunea vă prezintă modul de generare a secvențelor fără utilizarea unei coloane AUTO_INCREMENT.

AUTO_INCREMENT *pentru versiunile MySQL anterioare versiunii 3.23*

Pentru versiunile sistemului MySQL anterioare versiunii 3.23, coloanele AUTO_INCREMENT se comportă după cum urmează:

- Inserția unei valori NULL într-o coloană AUTO_INCREMENT determină MySQL să genereze automat următorul număr din secvență și să insereze în schimb acea valoare în coloană. Secvențele AUTO_INCREMENT încep cu 1, deci prima înregistrare inserată în tabel primește în coloană o valoare secvențială egală cu 1, iar înregistrările ulterioare primesc valori egale cu 2, 3 etc. În general, fiecare valoare generată automat va fi cu o unitate mai mare decât valoarea maximă curentă stocată în coloană.
- Inserția unui 0 într-o coloană AUTO_INCREMENT este similară cu inserția valorii NULL în coloană. Inserția unui rând fără specificarea unei valori pentru coloana AUTO_INCREMENT este de asemenea similară cu inserția unei valori NULL.
- Dacă inserați o înregistrare și specificați în mod explicit o valoare pentru coloana AUTO_INCREMENT, se va întâmpla unul din următoarele două lucruri. Dacă există deja o înregistrare cu valoarea respectivă, se va produce o eroare, deoarece valorile din coloanele AUTO_INCREMENT trebuie să fie unice. Dacă nu există o înregistrare cu acea valoare, înregistrarea este inserată și, dacă valoarea din coloană este noua valoare maximă, secvența continuă cu valoarea următoare aceleia pentru rândurile ulterioare. Cu alte cuvinte, puteți „forța" contorul prin inserția unei înregistrări cu o valoare din secvență mai mare decât valoarea curentă a contorului.

  Forțarea contorului poate determina goluri în secvență, dar puteți exploata această comportare în avantajul dumneavoastră. Să presupunem că ați creat un tabel cu o coloană AUTO_INCREMENT, dar doriți ca o secvență să înceapă cu 1000, nu cu 1. Puteți realiza acest lucru în două moduri. Mai întâi, puteți insera prima înregistrare cu o valoare de secvență explicită egală cu 1000, după care inserați înregistrările ulterioare prin introducerea valorii NULL în coloana AUTO_INCREMENT. În al doilea rând, puteți insera o falsă înregistrare în coloana AUTO_INCREMENT, cu valoarea 999. Prima înregistrare reală pe care o inserați după aceea va primi un număr de secvență egal cu 1000, după care puteți șterge înregistrarea falsă.

- Dacă inserați o valoare incorectă într-o coloană AUTO_INCREMENT, nu vă așteptați să se întâmple ceva util. Rezultatul nu poate fi anticipat.
- Dacă ștergeți înregistrarea care conține cea mai mare valoare dintr-o coloană AUTO_INCREMENT, valoarea respectivă va fi reutilizată la următoarea generare a unei valori noi. Dacă ștergeți toate înregistrările din tabel, vor fi refolosite toate valorile și secvența este reluată, pornind de la 1.
- Instrucțiunile REPLACE funcționează normal.
- Instrucțiunile UPDATE funcționează folosind reguli similare celor care se aplică inserției de noi înregistrări. Dacă actualizați o coloană AUTO_INCREMENT cu valoarea NULL sau 0, aceasta este actualizată la următorul număr de secvență. Dacă încercați să actualizați coloana la o valoare care există deja, se va produce o eroare (cu excepția situațiilor când se întâmplă să configurați coloana la o valoare pe care o are deja). Dacă actualizați coloana la o valoare mai mare decât orice valoare existentă în coloană, secvența va continua cu numărul următor celui pentru înregistrările ulterioare.
- Valoarea celui mai recent număr de secvență generat automat este disponibilă prin apelarea funcției LAST_INSERT_ID(). Aceasta vă permite să faceți referire la valoarea AUTO_INCREMENT în alte instrucțiuni, fără a cunoaște care este de fapt valoarea. LAST_INSERT_ID() este legată de valorile AUTO_INCREMENT generate în timpul sesiunii curente a serverului; nu este afectată de activitatea AUTO_INCREMENT asociată cu alți clienți. Dacă nici o valoare AUTO_INCREMENT nu fost generată în timpul sesiunii curente, funcția LAST_INSERT_ID() returnează 0.

Posibilitatea de a genera automat numerele dintr-o secvență este extrem de utilă. Totuși, comportarea descrisă anterior are două dezavantaje. Primul: reutilizarea valorilor dintr-o secvență atunci când sunt șterse înregistrările din partea superioară a secvenței îngreunează generarea unui set de valori monotone (strict crescătoare) pentru aplicații care pot șterge, dar care pot și insera înregistrări. În al doilea rând, mijloacele prin care începeți o secvență la o valoare mai mare decât 1 sunt greoaie.

AUTO_INCREMENT *pentru versiunile MySQL începând de la 3.23*

MySQL 3.23 a introdus următoarele schimbări în comportarea atributului AUTO_INCREMENT în ceea ce privește aspectele reținute anterior:

- Valorile dintr-o serie generată automat sunt strict crescătoare și nu sunt refolosite. Dacă valoarea maximă este 143 și dumneavoastră ștergeți înregistrarea care conține valoarea respectivă, MySQL va genera valoarea următoare egală cu 144.
- Puteți specifica numărul de secvență inițial în mod explicit atunci când creați tabelul. Exemplul următor creează un tabel cu o coloană AUTO_INCREMENT denumită sec, care începe de la valoarea 1.000.000:

```
CREATE TABLE tabelul_meu
    (sec INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY)
    AUTO_INCREMENT = 1000000
```

  Când un tabel are mai multe coloane (ca majoritatea tabelelor), nu există nici un dubiu cu privire la coloana căreia i se aplică clauza finală AUTO_INCREMENT = 1000000, deoarece într-un tabel există o singură coloană AUTO_INCREMENT.

Aspecte de reținut privind atributul AUTO_INCREMENT

Trebuie să rețineți următoarele elemente pentru a evita surprizele atunci când folosiți coloane AUTO_INCREMENT:

- AUTO_INCREMENT nu este un tip de coloană; este un atribut al unui tip de coloană. Mai departe, AUTO_INCREMENT este un atribut destinat a fi utilizat numai pentru tipuri întregi. Versiunile de MySQL anterioare versiunii 3.23 nu aplică această restricție cu strictețe și vă vor permite să declarați un tip de coloană cum ar fi CHAR cu atributul AUTO_INCREMENT. Totuși, numai tipurile întregi funcționează corect drept coloane AUTO_INCREMENT.
- Scopul principal al mecanismului AUTO_INCREMENT este acela de a vă permite să generați o secvență de întregi pozitivi; cel mai indicat este să folosiți coloanele AUTO_INCREMENT numai în acest mod. Din acest motiv, trebuie să declarați coloanele AUTO_INCREMENT ca fiind UNSIGNED, ceea ce are în plus și avantajul de a vă oferi de două ori mai multe numere în secvență înainte de a ajunge la limita superioară a domeniului tipului de coloană.

  În anumite circumstanțe, este posibil să generați secvențe de valori negative folosind o coloană AUTO_INCREMENT, dar nu vă recomand această operație. Dacă sunteți hotărât să o încercați, nu uitați să efectuați testări adecvate și să re-efectuați testările dacă treceți la o versiune mai recentă, diferită, de MySQL. Propriile mele experiențe au scos în evidență o comportare oarecum inconsecventă a versiunilor în ceea ce privește secvențele de numere negative.
- Nu vă lăsați amăgit de ideea că adăugarea atributului AUTO_INCREMENT la o declarație de coloană este o modalitate magică de a obține o secvență nelimitată de numere. Nici vorbă de așa ceva; secvențele AUTO_INCREMENT sunt limitate de domeniul tipului de coloană. De exemplu, dacă folosiți o coloană TINYINT UNSIGNED, numărul maxim de secvență este 255. Când ajungeți la această limită, aplicația dumneavoastră va începe să înregistreze erori de genul „cheie duplicată".
- MySQL 3.23 a introdus noile caracteristici ale atributului AUTO_INCREMENT de a nu refolosi numerele de secvență și de a vă permite să specificați un număr inițial de secvență în instrucțiunea CREATE TABLE. Aceste caracteristici sunt anulate dacă ștergeți toate înregistrările din tabel folosind o instrucțiune DELETE de forma următoare:

  ```
  DELETE FROM nume_tabel
  ```

  În acest caz, secvența va începe de la 1, în loc să continue într-o ordine strict crescătoare. Secvența reia numărătoarea de la 1 chiar dacă instrucțiunea dumneavoastră CREATE TABLE specifică în mod explicit un număr inițial de secvență. Acest lucru se produce datorită modului în care MySQL optimizează instrucțiunile DELETE care golesc în întregime un tabel: re-creează datele și fișierele index de la zero, în loc de a șterge fiecare înregistrare, ceea ce determină pierderea tuturor informațiilor despre numerele de secvență. Dacă doriți să ștergeți toate înregistrările, dar să păstrați informațiile de secvență, puteți suprima optimizarea și forțați MySQL să execute în schimb o operație de ștergere rând cu rând, astfel:

  ```
  DELETE FROM nume_tabel WHERE 1 > 0
  ```

Ce puteți face pentru a păstra o serie strict crescătoare dacă aveți o versiune de MySQL mai veche decât 3.23? O soluție este de a folosi un tabel separat, pe care îl folosiți numai pentru generarea de valori AUTO_INCREMENT și din care nu ștergeți niciodată înregistrări.

Astfel, valorile din tabel nu sunt refolosite niciodată. Când trebuie să generați o înregistrare nouă în tabelul dumneavoastră principal, mai întâi inserați o valoare NULL în tabelul cu numere de secvență. Apoi, inserați înregistrarea în tabelul dumneavoastră principal folosind valoarea funcției LAST_INSERT_ID() pentru coloana care doriți să conțină un număr de secvență:

```
INSERT INTO ai_tbl SET ai_col = NULL
INSERT INTO tbl_principal SET id=LAST_INSERT_ID()...
```

Să presupunem că doriți să scrieți o aplicație care generează valori AUTO_INCREMENT, dar doriți ca secvența să înceapă cu 100 în loc de 1. Să mai presupunem că doriți ca aplicația să fie portabilă la toate versiunile de MySQL. Cum puteți realiza acest deziderat?

Dacă doriți să realizați portabilitatea, nu puteți conta pe faptul că MySQL 3.23 asigură specificarea numărului de secvență inițial în instrucțiunea CREATE TABLE. În schimb, când doriți să inserați o interogare, mai întâi verificați dacă tabelul este gol sau nu prin emiterea următoarei instrucțiuni:

```
SELECT COUNT(*) FROM nume_tabel
```

Aceasta este o etapă suplimentară, dar care nu vă costă prea mult, deoarece instrucțiunea SELECT COUNT(*) fără nici o clauză WHERE este optimizată să se execute rapid. Dacă tabelul este vid, inserați înregistrarea și specificați în mod explicit valoarea 100 pentru coloana numerelor de secvență. Dacă tabelul nu este vid, pur și simplu specificați NULL pentru valoarea din coloana cu numărul de secvență și permiteți programului MySQL să genereze automat următorul număr.

Această metodă vă permite să inserați înregistrări cu numerele de secvență 100, 101 și așa mai departe și funcționează indiferent dacă MySQL permite sau nu specificarea valorii inițiale a secvenței. Această abordare nu funcționează dacă impuneți ca numerele de secvență să fie strict crescătoare chiar și atunci când sunt șterse înregistrări din tabel. În cazul respectiv, puteți combina această metodă cu tehnica descrisă anterior, de utilizare a unui tabel secundar strict pentru generarea numerelor de secvență care vor fi folosite în tabelul dumneavoastră principal.

De ce să începeți o secvență cu o valoare mai mare decât 1? Un motiv este de a determina toate numerele de secvență să aibă același număr de cifre. Dacă generați numere de identificare a clienților și nu vă așteptați la mai mult de un milion de clienți, puteți începe seria de la numărul 1.000.000. Veți putea insera cu mult peste un milion de înregistrări ale clienților înainte ca numărul de cifre al identificatorului de client să se modifice.

O altă modalitate de a forța o anumită dimensiune a numerelor de secvență este de a folosi o coloană ZEROFILL, desigur. Acest procedeu poate ridica probleme, în funcție de contextul în care vă folosiți datele. De exemplu, dacă manipulați numere de secvență cu zerouri inițiale în scripturi Perl sau PHP, va trebui să le folosiți numai ca șiruri; dacă vor fi convertite în numere, zerourile inițiale se vor pierde. Următorul script Perl scurt ilustrează pericolele lucrului cu numere de acest gen:

```
#! usr/bin/perl
$s = "00010";      # creeaza "numar" cu zerouri initiale
print "$s\n";
$s++;              # foloseste incrementul inteligent din Perl
```

```
print "$s\n";
$s += 1;          # foloseste $s intr-un context numeric
print "$s\n";
```

Când este executat, acest script afişează următoarele date de ieşire:

```
00010    OK
00011    OK
12       Hopa!
```

Operatorul ++ de auto-incrementare din Perl este inteligent şi poate crea valori secvenţiale din şiruri sau numere, dar operatorul += funcţionează numai cu numere. În datele de ieşire afişate mai sus, puteţi vedea că operatorul += cauzează o conversie şir-număr, iar zerourile iniţiale din valoarea lui `$s` se pierd.

Alte motive pentru a nu începe o secvenţă cu 1 s-ar putea să nu aibă nici o legătură cu anumite consideraţii de natură tehnică. De exemplu, dacă atribuiţi numere de membru, puteţi dori să începeţi o secvenţă la un număr mai mare decât 1, pentru a dejuca manevrele politice legate de identitatea membrului nr. 1 şi pentru a vă asigura că un asemenea număr nu există. Se mai întâmplă şi aşa ceva... Trist, dar adevărat.

Generarea secvenţelor fără AUTO_INCREMENT

O altă metodă de generare a numerelor secvenţiale nu foloseşte o coloană `AUTO_INCREMENT`. În schimb, foloseşte forma alternativă a funcţiei `LAST_INSERT_ID()`, care preia un argument. (Această formă a fost introdusă în MySQL 3.22.9.) Dacă inseraţi sau actualizaţi o coloană folosind `LAST_INSERT_ID(expr)`, următorul apel la `LAST_INSERT_ID()` fără nici un argument va returna valoarea `expr`. Cu alte cuvinte, `expr` este tratată ca şi cum ar fi fost generată de către mecanismul `AUTO_INCREMENT`. Aceasta vă permite să generaţi un număr de secvenţă şi apoi să-l folosiţi într-o instrucţiune ulterioară în cadrul sesiunii client, fără ca valoarea să fie afectată de alţi clienţi.

O modalitate de a folosi această strategie este de a crea un tabel cu un singur rând, conţinând o valoare care este actualizată de fiecare dată când aveţi nevoie de următoarea valoare din secvenţă. De exemplu, puteţi crea tabelul în acest mod:

```
CREATE TABLE sec_tabel (sec INT UNSIGNED NOT NULL)
INSERT INTO sec_tabel VALUES(0)
```

Aceste instrucţiuni creează tabelul `sec_tabel` şi îl iniţializează cu un singur rând, care conţine o valoare `sec` egală cu 0. Pentru a folosi tabelul, generaţi următorul număr din secvenţă, după cum urmează:

```
UPDATE sec_tabel SET sec = LAST_INSERT_ID(sec+1)
```

Această instrucţiune regăseşte valoarea curentă a coloanei `sec` şi o incrementează cu 1 pentru a genera următoarea valoare din secvenţă. Generarea noii valori folosind `LAST_INSERT_ID(sec+1)` determină tratarea acesteia ca şi cum ar fi fost o valoare `AUTO_INCREMENT`, iar valoarea poate fi regăsită printr-o instrucţiune ulterioară, apelând funcţia `LAST_INSERT_ID()` fără argument. Acest procedeu funcţionează chiar dacă un alt client a generat între timp un alt număr din secvenţă, deoarece `LAST_INSERT_ID()` este specific fiecărui client în parte.

De asemenea, puteţi folosi această metodă dacă doriţi să generaţi valori secvenţiale care se incrementează cu o valoare diferită de 1, respectiv valori secvenţiale negative. De exemplu, următoarele două instrucţiuni pot fi folosite pentru a genera o secvenţă de numere care cresc cu câte 100 de unităţi, respectiv o secvenţă de numere negative:

```
UPDATE sec_tabel SET sec = LAST_INSERT_ID(sec+100)
UPDATE sec_tabel SET sec = LAST_INSERT_ID(sec-1)
```

Mai puteţi folosi această metodă şi pentru generarea unei secvenţe care începe de la o valoare arbitrară prin stabilirea în coloana `sec` a unei valori iniţiale adecvate.

Pentru o aplicaţie a acestei metode de generare a secvenţelor pentru mai multe contoare, vezi „Configurarea unui tabel contor" în capitolul 3, „Sintaxa şi utilizarea SQL în MySQL".

Tipuri de coloană şir

MySQL furnizează numeroase tipuri de şiruri pentru stocarea datelor de tip caracter. Şirurile sunt frecvent folosite pentru valori ca acestea:

```
"Ionescu, Popescu & Co."
"Creioane (mina nr. 2)"
"Str. Lunga, nr. 123"
"Seria Monograph IX"
```

De fapt, şirurile sunt într-un fel tipuri generice, deoarece le puteţi folosi pentru a reprezenta orice valoare. De exemplu, puteţi folosi tipurile şir pentru a stoca date binare, cum sunt imaginile sau sunetele, respectiv date de ieşire din programul `gzip`, dacă doriţi să stocaţi date comprimate.

Pentru toate tipurile şir, valorile prea lungi sunt „tăiate" pentru a corespunde lungimii stabilite. Dar tipurile şir variază de la foarte mici la foarte mari, iar tipul cel mai mare poate conţine aproape 4GB de date, deci trebuie să puteţi găsi ceva suficient de lung pentru a evita trunchierea informaţiilor dumneavoastră[4].

Tabelul 2.8 prezintă tipurile furnizate de MySQL pentru declararea coloanelor cu valori de tip şir, precum şi dimensiunea maximă şi spaţiul de stocare necesar pentru fiecare tip. Pentru tipuri de coloane cu lungime variabilă, cantitatea de spaţiu ocupată de o valoare variază de la un rând la altul şi depinde de lungimea valorilor efectiv stocate în coloană. Această lungime este reprezentată în tabel prin litera `L`.

Octeţii suplimentari necesari în afara lui `L` reprezintă numărul de octeţi necesari pentru a stoca lungimea valorii. MySQL manipulează valorile de lungime variabilă prin stocarea atât a conţinutului valorii, cât şi a lungimii sale. Aceşti octeţi suplimentari sunt trataţi ca un întreg fără semn. Observaţi corespondenţa între lungimea maximă a unui tip de lungime variabilă, numărul de octeţi suplimentari necesari pentru tipul respectiv şi domeniul tipului întreg fără semn care foloseşte acelaşi număr de octeţi. De exemplu, valorile `MEDIUMBLOB` pot avea maximum 2^{24}-1 octeţi lungime şi necesită 3 octeţi pentru înregistrarea rezultatului. Tipul întreg pe 3 octeţi `MEDIUMINT` are o valoare fără semn maximă de 2^{24}-1. Aceasta nu este o coincidenţă.

[4] Datorită limitărilor impuse de dimensiunea maximă a pachetului în protocolul de comunicaţie client/server, limita eficace a valorilor din coloană este de 24MB. – N.A.

Tabelul 2.8 Tipuri de coloane şir.

Specificaţia tipului	Dimensiune maximă	Spaţiu necesar
`CHAR(M)`	*M* octeţi	*M* octeţi
`VARCHAR(M)`	*M* octeţi	*L* + 1 octeţi
`TINYBLOB, TINYTEXT`	2^8-1 octeţi	*L* + 1 octeţi
`BLOB, TEXT`	2^{16}-1 octeţi	*L* + 2 octeţi
`MEDIUMBLOB, MEDIUMTEXT`	2^{24}-1 octeţi	*L* + 3 octeţi
`LONGBLOB, LONGTEXT`	2^{32}-1 octeţi	*L* + 4 octeţi
`ENUM ("val1","val2",...)`	65535 membri	1 sau 2 octeţi
`SET ("val1","val2",...)`	64 membri	1, 2, 3, 4 sau 8 octeţi

Tipurile de coloane CHAR şi VARCHAR

CHAR şi VARCHAR sunt cele mai folosite tipuri şir. Diferenţa dintre ele constă în aceea că CHAR este un tip de lungime fixă, iar VARCHAR este un tip de lungime variabilă. Valorile dintr-o coloană CHAR(M) ocupă fiecare câte M octeţi; valorile mai scurte sunt completate la dreapta cu spaţii atunci când sunt stocate. (Totuşi, la regăsirea acestor valori, spaţiile finale sunt eliminate.) Valorile dintr-o coloană VARCHAR(M)sunt stocate folosind numai cantitatea de octeţi necesară, plus un octet pentru înregistrarea lungimii[5].

Dacă lungimea valorilor dumneavoastră nu variază prea mult, CHAR este o opţiune mai bună decât VARCHAR, deoarece tabelele cu rânduri de lungime fixă pot fi prelucrate mai eficient decât tabelele cu lungime variabilă. Dacă valorile dumneavoastră au toate aceeaşi lungime, VARCHAR va folosi, de fapt, un spaţiu mai mare, datorită octetului suplimentar utilizat pentru înregistrarea lungimii valorilor.

Anterior versiunii MySQL 3.23, coloanele CHAR şi VARCHAR pot fi declarate cu o lungime maximă M cuprinsă între 1 şi 255. Începând de la versiunea MySQL 3.23, CHAR(0) este de asemenea o valoare permisă. CHAR(0) este util ca element de înlocuire atunci când doriţi să declaraţi o coloană, dar nu vreţi să alocaţi spaţiu pentru aceasta dacă nu sunteţi sigur de lăţimea pe care doriţi să i-o atribuiţi. Puteţi folosi ALTER TABLE pentru a mări ulterior lăţimea coloanei. O coloană CHAR(0) poate fi de asemenea folosită pentru a reprezenta valorile pornit/oprit, dacă îi permiteţi să ia valoarea NULL. Valorile dintr-o asemenea coloană pot lua două valori, în speţă NULL şi şirul vid. O coloană CHAR(0) ocupă un spaţiu de stocare foarte redus în interiorul tabelului: un singur bit.

Cu anumite excepţii limitate, nu puteţi combina CHAR şi VARCHAR în interiorul aceluiaşi tabel. MySQL va proceda chiar la modificarea coloanelor de la un tip la altul, în funcţie de circumstanţe. (Acesta este un lucru care nu se întâmplă în cadrul altor sisteme de baze de date.) Principiile care se aplică sunt următoarele:

- Tabelele cu rânduri de lungime fixă sunt prelucrate mai uşor decât tabelele cu rânduri de lungime variabilă. (Motivele vor fi discutate în secţiunea „Alegerea tipurilor de coloane".)

[5] Spaţiile finale sunt eliminate la stocarea valorilor; acest procedeu diferă de standardul ANSI pentru SQL în ceea ce priveşte valorile de tip VARCHAR. – N.A.

- Rândurile unui tabel au lungime fixă numai dacă toate coloanele din tabel sunt tipuri cu lungime fixă. Dacă fie şi o singură coloană are lungime variabilă, rândurile tabelului devin şi ele de lungime variabilă.
- Deoarece avantajele de performanţă ale rândurilor de lungime fixă se pierd atunci când rândul devine de lungime variabilă, toate coloanele de lungime fixă pot fi convertite în echivalente de lungime variabilă, atunci când o atare măsură poate duce la economii de spaţiu.

Aceasta înseamnă că, dacă aveţi coloane VARCHAR într-un tabel, nu puteţi avea *şi* coloane CHAR; MySQL le converteşte „pe tăcute" în VARCHAR. Să presupunem că aţi creat un tabel ca acesta:

```
CREATE TABLE tabelul_meu
(
    c1 CHAR(10),
    c2 VARCHAR(10)
)
```

Dacă emiteţi o interogare DESCRIBE tabelul_meu, datele de ieşire se prezintă astfel:

Field	Type	Null	Key	Default	Extra
c1	varchar(10)	YES		NULL	
c2	varchar(10)	YES		NULL	

Observaţi că prezenţa coloanei VARCHAR determină programul MySQL să convertească şi coloana c1 la tipul VARCHAR. Dacă încercaţi să folosiţi ALTER TABLE pentru a converti c1 la CHAR, nu veţi reuşi. Unica modalitate de a transforma o coloană VARCHAR în coloană CHAR este de a converti toate coloanele VARCHAR din tabel în acelaşi timp:

```
ALTER TABLE tabelul_meu MODIFY c1 CHAR(10), MODIFY c2 CHAR(10)
```

Tipurile de coloane BLOB şi TEXT sunt de lungime variabilă, ca şi VARCHAR, dar nu au nici un echivalent de lungime fixă, deci nu puteţi folosi coloane CHAR în acelaşi tabel ca şi coloanele BLOB şi TEXT. Orice coloană CHAR va fi convertită la VARCHAR.

Excepţia de la regula neutilizării în acelaşi tabel a coloanelor de lungime fixă şi a celor de lungime variabilă constă în aceea că acele coloane de tip CHAR mai scurte de patru caractere nu sunt convertite la VARCHAR. De exemplu, MySQL nu va transforma coloana CHAR din următorul tabel în VARCHAR:

```
CREATE TABLE tabelul_meu
(
    c1 CHAR(2),
    c2 VARCHAR(10)
)
```

Motivul pentru care coloanele mai scurte de patru caractere nu sunt convertite este acela că, în medie, economia de spaţiu pe care o puteţi obţine prin nestocarea spaţiilor finale este anulată de octetul suplimentar necesar într-o coloană VARCHAR pentru înregistrarea lungimii fiecărei valori. De fapt, dacă *toate* coloanele dumneavoastră sunt scurte, MySQL va converti la CHAR orice coloană declarată ca VARCHAR. MySQL procedează astfel deoarece conversia nu va mări, în medie, cantitatea de spaţiu necesară şi va îmbunătăţi

performanțele prin transformarea rândurilor tabelului în rânduri de lungime fixă. Dacă creați un tabel cu următoarea specificație, coloanele VARCHAR vor fi toate convertite „discret" în CHAR:

```
CREATE TABLE tabelul_meu
(
    c1 VARCHAR(1),
    c2 VARCHAR(2),
    c3 VARCHAR(3)
)
```

Puteți verifica modificarea coloanelor prin examinarea datelor de ieșire ale interogării DESCRIBE tabelul_meu:

Field	Type	Null	Key	Default	Extra
c1	char(1)	YES		NULL	
c2	char(2)	YES		NULL	
c3	char(3)	YES		NULL	

Tipurile de coloane BLOB și TEXT

Un "BLOB" este un obiect binar mare (*b*inary *l*arge *ob*ject), în esență un container care poate stoca orice doriți să puneți în el și pe care îl puteți face aproape oricât de mare doriți. În MySQL, tipul BLOB este de fapt o familie de tipuri (TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB) care sunt identice, cu excepția cantității maxime de informații pe care o pot stoca (vezi tabelul 2.8). MySQL mai are o familie de tipuri TEXT (TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT). Acestea sunt identice din toate punctele de vedere cu tipurile BLOB corespunzătoare, cu excepția faptului că, la comparare și sortare, valorile BLOB sunt sensibile la diferența între majuscule și minuscule, în timp ce valorile TEXT nu sesizează această diferență. Coloanele BLOB și TEXT sunt utile pentru stocarea datelor care pot deveni foarte mari sau care pot varia foarte mult ca dimensiune de la un rând la altul. Printre exemple se numără documentele create cu procesoarele de text, imaginile și sunetele, datele compuse și articolele de știri.

Coloanele BLOB și TEXT pot fi indexate începând de la MySQL 3.23.2, deși trebuie să specificați o dimensiune a prefixului care va fi folosit pentru index, cu scopul de a evita intrările de index care pot deveni enorme și care pot, ca atare, să anuleze toate avantajele dobândite prin utilizarea acelui index. De asemenea, în general nu se efectuează căutări în coloane BLOB sau TEXT, deoarece coloane ca acestea conțin deseori date binare (precum imaginile). Se obișnuiește mai frecvent să se utilizeze alte coloane din tabel pentru înregistrarea unui anumit tip de informații de identificare referitoare la valorile BLOB sau TEXT și să se folosească acele informații pentru a determina rândurile dorite.

Coloanele BLOB și TEXT pot necesita o atenție specială:

- Datorită variațiilor mari caracteristice ale dimensiunilor coloanelor BLOB și TEXT, tabelele care le conțin sunt supuse unor fragmentări de proporții dacă sunt efectuate numeroase ștergeri și actualizări. Va fi necesar să rulați periodic instrucțiunea OPTIMIZE TABLE pentru a reduce fragmentarea și pentru a păstra un nivel ridicat de performanță. Vezi capitolul 4, „Optimizarea interogărilor", pentru mai multe informații.

- Dacă folosiți valori foarte mari, poate fi necesar să ajustați serverul în vederea creșterii valorii parametrului max_allowed_packet. Vezi capitolul 11, „Administrarea generală a sistemului MySQL", pentru mai multe informații. De asemenea, va trebui să măriți dimensiunea pachetului pentru orice client care dorește să folosească valori foarte mari. Anexa E, „Referință de programe MySQL", descrie această operație pentru clienții mysql și mysqldump.

Tipurile de coloane ENUM și SET

ENUM și SET sunt tipuri de șiruri speciale pentru care valorile coloanelor trebuie alese dintr-un set fix de șiruri. Principala diferență dintre cele două tipuri este că valorile coloanelor de tip ENUM trebuie să fie alcătuite dintr-un singur membru al setului de valori, în timp ce valorile dintr-o coloană SET pot conține oricare membru al setului sau chiar pe toți membrii. Cu alte cuvinte, ENUM este folosită pentru valori mutual exclusive, în timp ce SET permite mai multe opțiuni dintr-o listă de valori.

Tipul de coloană ENUM definește o enumerare. În coloanele ENUM se pot repartiza valori alcătuite din exact un membru ales dintr-o listă de valori specificată în momentul creării tabelului. O enumerare poate avea maximum 65536 membri (din care unul este rezervat de către MySQL). Enumerările se folosesc frecvent pentru a reprezenta valorile dintr-o categorie. De exemplu, valorile dintr-o coloană declarată ca ENUM("N","D") pot fi numai "N" sau "D". Alternativ, puteți folosi ENUM pentru răspunsuri la întrebări cu opțiuni multiple dintr-un chestionar sau studiu, respectiv pentru mărimile și culorile unui produs:

```
salariati ENUM("sub 100","100-500","501-1500","peste 1500")
culoare ENUM("rosu","verde","albastru","negru")
marime ENUM("S","M","L","XL","XXL")
```

Dacă prelucrați selecții din pagini Web, puteți folosi ENUM pentru a reprezenta opțiunea pe care un vizitator al sitului dumneavoastră o alege dintr-un set de butoane radio mutual exclusive plasate într-o pagină. De exemplu, dacă aveți un serviciu pe Internet pentru comenzi de pizza, se poate folosi o enumerare pentru a reprezenta tipul de coajă comandat de un client:

```
coaja ENUM("subtire","normala","stil pan")
```

În cazul în care categoriile de enumerare reprezintă numere, este important să vă alegeți categoriile în mod corespunzător atunci când creați enumerarea. De exemplu, când înregistrați numărul de leucocite rezultat în urma unui test de laborator, puteți grupa numerele în categorii în acest mod:

```
leucocite ENUM("0-100","101-300",">300")
```

Când rezultatul testului este dat sub forma unei valori exacte, puteți înregistra valoarea sub forma categoriei în care se încadrează aceasta. Nu puteți însă reveni la valoarea originală dacă decideți să convertiți coloana dintr-o enumerare bazată pe categorii într-o coloană de întregi, compusă din valori exacte.

Tipul SET este similar cu ENUM în sensul că, atunci când creați o coloană SET, specificați o listă cu membrii permiși ai setului. Dar, spre deosebire de ENUM, fiecare valoare a coloanei poate fi alcătuită din orice număr de membri ai setului. Setul poate avea maximum 64 de membri. Puteți folosi un SET atunci când aveți un set fix de valori care nu sunt mutual exclusive, așa cum este cazul în coloana ENUM. De exemplu, puteți folosi SET pentru a reprezenta opțiunile disponibile pentru un autoturism:

```
SET("portbagaj","pilot automat","aer conditionat","trapa")
```

Apoi, valorile particulare din SET vor reprezenta opțiunile efectiv comandate de clienți:

```
SET("pilot automat,trapa")
SET("portbagaj,aer conditionat")
SET("portbagaj,pilot automat,aer conditionat")
SET("aer conditionat")
SET("")
```

Șirul vid indică faptul că un client nu a comandat nici o opțiune. Aceasta este o valoare admisă pentru SET.

Valorile din coloana SET sunt reprezentate sub forma unui șir unic. Dacă o valoare este alcătuită din mai mulți membri ai unui set, membrii sunt separați în interiorul șirului prin virgule. Evident, aceasta înseamnă că nu trebuie să folosiți un șir care include o virgulă ca membru al unei coloane SET.

Coloanele SET mai pot fi utilizate pentru reprezentarea unor informații precum diagnosticele unor pacienți sau rezultate din selecții efectuate în pagini Web. Pentru un diagnostic, poate exista o listă standard de simptome a căror (in)existență trebuie aflată de la pacient, iar acesta poate prezenta oricare simptom sau pe toate. Pentru serviciul dumneavoastră de comenzi de pizza prin Internet, pagina Web pentru comenzi poate avea un set de casete de validare pentru ingredientele din pizza pe care le dorește clientul, ingrediente din care se pot alege mai multe.

Modul în care declarați lista cu valori admise pentru o coloană ENUM sau SET este important din mai multe puncte de vedere:

- Lista determină valorile admise posibile care pot fi incluse în coloană, așa cum s-a arătat anterior.
- Puteți insera valori ENUM și SET folosind atât majuscule, cât și minuscule, dar mărimea literelor pentru șirurile specificate în declarația coloanei determină mărimea literei valorilor coloanei atunci când acestea sunt regăsite ulterior. De exemplu, dacă aveți o coloană ENUM("D","N") și stocați în coloană valorile "d" și "n", valorile sunt afișate sub forma "D" și "N" atunci când le regăsiți. Acest fapt nu afectează comportarea tipurilor respective la comparație sau sortare, deoarece coloanele ENUM și SET nu sunt sensibile la diferența între majuscule și minuscule.
- Ordinea valorilor dintr-o declarație ENUM este ordinea folosită pentru sortare. Ordinea valorilor dintr-o declarație SET determină de asemenea ordinea de sortare, deși relația este mai complicată în acest caz, deoarece valorile coloanelor pot conține mai mulți membri ai setului.
- Ordinea valorilor dintr-o declarație SET determină ordinea în care apar sub-șirurile atunci când sunt afișate valorile din coloana SET care sunt alcătuite din mai mulți membri ai unui set.

ENUM și SET sunt clasificate ca tipuri șir, deoarece membrii enumerării, respectiv ai setului, sunt specificați sub formă de șir atunci când creați coloane de aceste tipuri. Totuși, membrii sunt stocați intern sub formă de numere și puteți lucra cu ei ca atare. Aceasta înseamnă că tipurile ENUM și SET sunt mai eficiente decât alte tipuri șir, deoarece pot fi

manipulate frecvent folosind operații numerice în locul operațiilor cu șiruri. De asemenea, înseamnă că valorile ENUM și SET pot fi folosite atât în contexte numerice, cât și în contexte cu șiruri.

Membrii ENUM din declarația coloanei sunt numerotați secvențial, începând de la 1. (0 este folosit de MySQL pentru membrul „eroare“, care este reprezentat sub formă de șir prin șirul vid.) Numărul valorilor dintr-o enumerare determină dimensiunea spațiului de stocare al unei coloane ENUM. Un octet poate reprezenta 256 de valori, doi octeți pot reprezenta 65536 de valori. (Comparați aceste valori cu domeniile tipurilor întregi pe un octet și pe doi octeți TINYINT UNSIGNED și SMALLINT UNSIGNED.) Astfel, numărul maxim de membri ai unei enumerări este 65536 (inclusiv membrul „eroare“), iar dimensiunea spațiului de stocare depinde de existența sau nu a unui număr de membri mai mare de 256. Puteți specifica un număr maxim de 65535 (nu 65536) de membri în declarația ENUM, deoarece MySQL rezervă un spațiu pentru membrul „eroare“ ca membru implicit al fiecărei enumerări. Când atribuiți o valoare incorectă într-o coloană ENUM, MySQL repartizează în schimb membrul „eroare“.

Iată un exemplu pe care îl puteți încerca folosind clientul mysql. Exemplul prezintă ordonarea numerică a membrilor unei enumerări și demonstrează, de asemenea, că valoarea NULL nu are nici un număr în cadrul ordonării:

```
mysql> CREATE TABLE e_tabel (e ENUM("ana","ion","nae","ela"));
mysql> INSERT INTO e_tabel
VALUES("ana"),("ion"),("nae"),("ela"),(""),(NULL);
mysql> SELECT e, e+0, e+1, e*3 FROM e_tabel;
```

e	e+0	e+1	e*3
ana	1	2	3
ion	2	3	6
nae	3	4	9
ela	4	5	12
	0	1	0
NULL	NULL	NULL	NULL

Puteți efectua operații cu membrii enumerării în funcție de nume sau de număr:

```
mysql> SELECT e FROM e_tabel WHERE e="nae";
```

e
nae

```
mysql> SELECT e FROM e_tabel WHERE e=3;
```

e
nae

Se poate declara șirul vid ca membru permis al unei enumerări. Acestui șir i se va atribui o valoare numerică diferită de zero, ca oricărui alt membru menționat în declarație. Totuși, utilizarea unui șir vid poate provoca oarecare derută, deoarece șirul respectiv este de asemenea folosit pentru membrul „eroare“, a cărui valoare numerică este 0. În exemplul următor, repartizarea valorii de enumerare incorecte "x" în coloana ENUM determină

repartizarea membrului de eroare. Acesta poate fi difererențiat de membrul șir vid numai atunci când este regăsit în formă numerică:

```
mysql> CREATE TABLE t (e ENUM("a","","b"));
mysql> INSERT INTO t VALUES("a"),(""),("b"),("x");
mysql> SELECT e e+0 FROM t;
```

e	e+0
a	1
	2
b	3
	0

Reprezentarea numerică a coloanelor SET este puțin diferită de aceea a coloanelor ENUM. Membrii unui set nu sunt numerotați secvențial. În schimb, fiecărui membru îi corespunde unui bit individual în valoarea SET. Primul membru al setului corespunde bitului 0, al doilea membru corespunde bitului 1 etc. O valoare SET numerică egală cu 0 corespunde șirului vid. Membrii SET sunt stocați ca valori pe biți. În acest mod se pot stoca în fiecare octet câte opt valori ale unui set, deci dimensiunea spațiului de stocare a unei coloane SET este determinată de numărul de membri ai setului, până la o valoare maximă de 64 de membri. Valorile SET pot ocupa 1, 2, 3, 4 sau 8 octeți pentru dimensiuni ale setului cuprinse respectiv între 1-8, 9-16, 17-24, 25-32 și 33-64.

Reprezentarea unei valori SET sub forma unui set de biți este cea care îi permite unei asemenea valori să fie alcătuită din mai mulți membri ai unui set. Într-o valoare se poate stabili orice combinație de biți, deci valoarea poate fi alcătuită din orice combinație de șiruri din declarația SET care corespund acestor biți.

Iată un exemplu care ilustrează relația dintre forma șir și forma numerică a unei coloane SET; valoarea numerică este afișată atât în formă zecimală, cât și în formă binară:

```
mysql> CREATE TABLE s_tabel (s SET("ana","ion","nae","ela"));
mysql> INSERT INTO s_tabel
VALUES("ana"),("ion"),("nae"),("ela"),(""),(NULL);
mysql> SELECT s, s+0, BIN(s+0) FROM s_tabel;
```

s	s+0	BIN(s+0)
ana	1	1
ion	2	10
nae	4	100
ela	8	1000
	0	0
NULL	NULL	NULL

Dacă repartizați într-o coloană SET o valoare care conține sub-șiruri ce nu au fost menționate ca membri ai setului, aceste șiruri sunt eliminate și în coloană este repartizată o valoare care constă din sub-șirurile rămase. Când repartizați valori în coloanele SET, nu este necesar ca sub-șirurile să fie menționate în aceeași ordine pe care ați folosit-o atunci când ați declarat coloana. Totuși, când regăsiți valoarea ulterior, membrii vor fi menționați în ordinea de declarare. Să presupunem că specificați o coloană SET pentru a reprezenta articole de mobilier, folosind următoarea declarație:


```
SET("masa","lampa","scaun")
```

Dacă repartizați în această coloană valoarea "scaun,canapea,masa", se vor întâmpla două lucruri. Primul: șirul "canapea" dispare, deoarece nu este un membru al setului. Al doilea: când regăsiți valoarea ulterior, aceasta apare sub forma "masa,scaun". Acest lucru se produce deoarece MySQL determină biții care corespund fiecărui sub-șir al valorii care trebuie repartizate și îi activează în cadrul valorii stocate. Sub-șirul "canapea" nu corespunde nici unui bit și este ignorat. La regăsire, MySQL construiește valoarea șirului din valoarea numerică prin parcurgerea biților în ordine, fapt care are ca efect reordonarea automată a sub-șirurilor în ordinea folosită la declararea coloanei. Această caracteristică mai are și următoarea semnificație: dacă specificați un membru al setului de mai multe ori într-o valoare, acesta va apărea o singură dată la regăsirea valorii. Dacă repartizați valoarea "lampa,lampa,lampa" într-o coloană SET, la regăsire aceasta va avea numai valoarea "lampa".

Faptul că MySQL modifică ordinea membrilor dintr-o valoare SET înseamnă că, în cazul în care căutați valori folosind un șir, trebuie să enumerați membrii în ordinea adecvată. Dacă inserați "scaun,masa" și apoi căutați "scaun,masa", nu veți găsi nimic; trebuie să căutați valoarea "masa,scaun".

Sortarea și indexarea coloanelor ENUM și SET se realizează în conformitate cu valorile interne (numerice) ale valorilor din coloană. Exemplul următor ar putea părea altfel incorect, deoarece valorile nu sunt sortate în ordine alfanumerică:

```
mysql> SELECT e FROM e_tabel ORDER BY e;
```

e
NULL
ana
ion
nae
ela

Valoarea NULL apare după sortare înaintea altor valori (sau după acestea, la sortarea în ordine descendentă).

Puteți exploata sortarea ENUM dacă aveți un set fixat de valori și doriți să le sortați într-o anumită ordine. Atribuiți coloanei tipul ENUM când creați tabelul și menționați valorile din enumerare în declarația coloanei, în ordinea în care doriți să fie sortate.

Pentru situații în care doriți ca o coloană ENUM să fie sortată în ordine lexicografică normală, puteți converti coloana într-un șir non-ENUM folosind CONCAT() și sortând rezultatul:

```
mysql> SELECT CONCAT(e) AS e_str FROM e_tabel ORDER BY e_str;
```

e_str
NULL
ana
ela
ion
nae

Atributele coloanelor de tip şir

Atributul BINARY poate fi specificat pentru tipurile CHAR şi VARCHAR pentru a determina tratarea valorilor din coloană sub formă de şiruri binare (adică sensibile la diferenţa între majuscule şi minuscule în cazul operaţiunilor de comparare şi sortare).

Atributele generale NULL sau NOT NULL pot fi specificate pentru oricare dintre tipurile şir. Dacă nu specificaţi nici unul din acestea, atributul prestabilit este NULL. Totuşi, declararea unei coloane de tip şir ca NOT NULL nu împiedică introducerea unui şir vid. O valoare vidă este diferită de o valoare inexistentă, deci nu faceţi greşeala de a crede că puteţi forţa o coloană şir să conţină valori nevide declarând-o NOT NULL. În cazul în care cereţi ca valorile şirurilor să fie nevide, aceasta este o restricţie pe care trebuie să o impuneţi în cadrul propriilor dumneavoastră aplicaţii.

De asemenea, puteţi specifica o valoare prestabilită folosind atributul DEFAULT pentru toate coloanele de tip şir, cu excepţia tipurilor BLOB şi TEXT. Dacă nu specificaţi o valoare prestabilită, o asemenea valoare va fi aleasă în mod automat. Pentru coloanele care pot conţine NULL, valoarea prestabilită este NULL. Pentru coloane care nu pot conţine NULL, valoarea prestabilită este şirul vid, cu excepţia tipului ENUM, unde valoarea prestabilită este primul membru al enumerării. (Pentru SET, valoarea prestabilită într-o coloană care nu poate conţine NULL este de fapt setul vid, dar acesta este echivalent cu şirul vid.)

Coloane de tip dată şi oră

MySQL furnizează numeroase tipuri de coloane pentru valorile temporale: DATE, DATETIME, TIME, TIMESTAMP şi YEAR. Tabelul 2.9 prezintă tipurile furnizate de MySQL pentru declararea coloanelor care conţin valori pentru dată şi oră, precum şi domeniul valorilor permise pentru fiecare tip. Tipul YEAR a fost introdus în MySQL versiunea 3.22. Celelalte tipuri au fost prezente în toate versiunile MySQL. Dimensiunile necesare ale spaţiului de stocare pentru fiecare tip sunt prezentate în tabelul 2.10.

Tabelul 2.9 Coloane de tip dată şi oră

Specificaţia tipului	Domeniu
DATE	între "1000-01-01" şi "9999-12-31"
TIME	între "-838:59:59" şi "838:59:59"
DATETIME	între "1000-01-01 00:00:00" şi "9999-12-31 23:59:59"
TIMESTAMP[(*M*)]	între 19700101000000 şi undeva în anul 2037
YEAR[(*M*)]	între 1901 şi 2155

Tabel 2.10 Dimensiunile spaţiului de stocare necesare pentru coloanele de tip dată şi oră

Specificaţia tipului	Dimensiuni necesare ale spaţiului de stocare
DATE	3 octeţi (anterior versiunii MySQL 3.22, 4 octeţi)
TIME	3 octeţi
DATETIME	8 octeţi
TIMESTAMP	4 octeţi
YEAR	1 octet

Fiecare tip dată şi oră are o valoare "zero" care este stocată atunci când inseraţi o valoare care este incorectă pentru tipul respectiv, aşa cum se poate vedea în tabelul 2.11. Această valoare este, de asemenea, valoarea prestabilită pentru coloanele de tip dată şi oră care sunt declarate NOT NULL.

Tabelul 2.11 Valorile „zero" pentru coloanele de tip dată şi oră

Specificaţia tipului	Valoare „zero"
DATE	"0000-00-00"
TIME	"00:00:00"
DATETIME	"0000-00-00 00:00:00"
TIMESTAMP	00000000000000
YEAR	0000

MySQL reprezintă întotdeauna datele începând cu anul, în conformitate cu specificaţia ANSI. De exemplu, 3 decembrie 1999 este reprezentat sub forma "1999-12-03". MySQL este oarecum lejer privind modul în care permite specificarea datelor de intrare. De exemplu, va converti valorile anilor compuse din două cifre în valori din patru cifre, iar dumneavoastră nu trebuie să furnizaţi un zero iniţial pentru valorile lunilor şi zilelor care sunt mai mici decât 10. Totuşi, trebuie să specificaţi mai întâi anul. Formatele cu care sunteţi mai obişnuit, precum "12/3/99" sau "3/12/99" vor fi interpretate incorect. Regulile folosite de MySQL pentru interpretarea datelor sunt discutate în detaliu în paragraful „Lucrul cu coloanele de tip dată şi oră".

Valorile de tip oră sunt returnate conform fusului orar local al serverului; MySQL nu execută nici un fel de modificări în funcţie de fusul orar ale valorilor pe care le returnează clientului.

Tipurile de coloane DATE, TIME şi DATETIME

Tipurile DATE, TIME şi DATETIME conţin valori de date, ore şi valori combinate de tip dată-oră. Formatele sunt "AAAA-LL-ZZ", "hh:mm:ss", "AAAA-LL-ZZ hh:mm:ss". Pentru tipul DATETIME, partea de dată şi cea de oră sunt obligatorii; dacă atribuiţi o valoare DATE unei coloane DATETIME, MySQL adaugă automat o parte de oră sub forma "00:00:00".

MySQL tratează ora din valorile DATETIME şi TIME în moduri uşor diferite. Pentru DATETIME, partea de oră reprezintă o oră din zi. Pe de altă parte, o valoare TIME reprezintă timpul scurs (iată de ce domeniul este atât de mare şi de ce sunt permise şi valorile negative). Partea din extremitatea dreaptă a valorii este considerată ca indicând secunde deci, dacă inseraţi o valoare de oră „scurtă" (incomplet definită), precum "12:30" într-o coloană TIME, valoarea stocată este "00:12:30", ceea ce este interpretat sub forma „12 minute, 30 de secunde". Puteţi folosi coloanele de tip TIME pentru a reprezenta ora din zi dacă doriţi, dar nu uitaţi de această regulă de conversie dacă doriţi să evitaţi problemele. Pentru a insera valoarea „12 ore, 30 de minute", trebuie să o specificaţi sub forma "12:30:00".

Tipul de coloană TIMESTAMP

Coloanele TIMESTAMP reprezintă valorile în formatul AAAALLZZhhmmss, cu un domeniu cuprins între 19700101000000 și undeva în anul 2037. Domeniul este legat de ora UNIX, unde prima zi din 1970 este „ziua zero", cunoscută și sub numele de „epocă". Începutul lui 1970 determină extremitatea inferioară a domeniului TIMESTAMP. Extremitatea superioară a domeniului corespunde limitei de 4 octeți a orei UNIX, care poate reprezenta valori în anul 2037[6].

Tipul TIMESTAMP are această denumire (amprentă de timp - N.T.) deoarece are proprietatea specială de a înregistra momentul creării sau al modificării unei înregistrări. Dacă inserați o valoare NULL într-o coloană TIMESTAMP, valoarea coloanei devine automat aceea a datei și orei curente. Acest lucru se întâmplă și atunci când creați sau actualizați un rând, dar nu atribuiți coloanei nici o valoare explicită. Totuși, numai prima coloană TIMESTAMP dintr-un rând este tratată în acest mod și, chiar și pentru prima coloană, puteți anula aplicarea amprentei de timp prin inserția în coloană a unei date și a unei ore explicite în locul valorii NULL.

Declarația unei coloane TIMESTAMP poate include o specificație pentru o lățime maximă de afișare M. Tabelul 2.12 afișează formatele de afișare pentru valorile permise ale lui M. Dacă M este omis dintr-o declarație TIMESTAMP sau are o valoare egală cu 0 sau mai mare decât 14, coloana este tratată ca TIMESTAMP(14). Valorile impare ale lui M cuprinse în intervalul 1-13 sunt asimilate următorului număr par imediat superior.

Tabelul 2.12 Formate de afișare pentru valorile TIMESTAMP

Specificația tipului	Formatul de afișare
TIMESTAMP(14)	*AAAALLZZhhmmss*
TIMESTAMP(12)	*AAAALLZZhhmm*
TIMESTAMP(10)	*AALLZZhhmm*
TIMESTAMP(8)	*AAAALLZZ*
TIMESTAMP(6)	*AALLZZ*
TIMESTAMP(4)	*AALL*
TIMESTAMP(2)	*AA*

Lățimea de afișare pentru coloanele TIMESTAMP nu are nici o legătură cu dimensiunea spațiului de stocare sau cu valorile stocate intern. Valorile TIMESTAMP sunt întotdeauna stocate sub formă de 4 octeți și sunt folosite în calcule cu precizie completă de 14 cifre, indiferent de lățimea de afișare. Pentru a vedea acest lucru, să presupunem că declarați un tabel după cum urmează, după care inserați câteva rânduri în tabel și le regăsiți:

[6] Limita superioară a valorilor TIMESTAMP va crește pe măsură ce sistemele de operare vor fi modificate pentru a extinde limita superioară a valorilor orei UNIX. Acesta este un aspect care trebuie abordat la nivelul bibliotecii de sistem. MySQL va exploata aceste modificări pe măsură ce acestea sunt efectuate. – N.A.


```
CREATE TABLE tabelul_meu
(
    ts TIMESTAMP(8),
    i INT
)
INSERT INTO tabelul_meu VALUES(19990801120000,3)
INSERT INTO tabelul_meu VALUES(19990801120001,2)
INSERT INTO tabelul_meu VALUES(19990801120002,1)
INSERT INTO tabelul_meu VALUES(19990801120003,0)
SELECT * FROM tabelul_meu ORDER BY ts, i
```

Datele de ieșire ale instrucțiunii SELECT se prezintă astfel:

ts	i
19990801	3
19990801	2
19990801	1
19990801	0

În aparență, rândurile sunt stocate într-o ordine greșită – valorile din prima coloană sunt toate identice, deci se pare că sortarea ar trebui să aranjeze rândurile în conformitate cu valorile din a doua coloană. Acest rezultat aparent anormal se datorează faptului că MySQL sortează în funcție de valorile complete, cu 14 cifre, inserate în coloana TIMESTAMP.

MySQL nu are nici un tip de coloane care să primească data și ora creării înregistrării și care să rămână imuabil după aceea. Dacă doriți să realizați acest lucru, puteți proceda în două moduri:

- Folosiți o coloană TIMESTAMP. Când înregistrarea este creată pentru prima dată, atribuiți coloanei valoarea NULL pentru a o inițializa cu valoarea datei și a orei curente:

  ```
  INSERT INTO nume_tabel (ts_col, ...) VALUES(NULL, ...)
  ```

  De fiecare dată când actualizați înregistrarea la o dată ulterioară, atribuiți în mod explicit coloanei valoarea pe care o are deja. Prin atribuirea unei valori explicite se contracarează mecanismul de aplicare a amprentei de timp, deoarece împiedică actualizarea automată a valorii coloanei:

  ```
  UPDATE nume_tabel SET ts_col=ts_col WHERE ...
  ```

- Folosiți o coloană DATETIME. Când creați o înregistrare, inițializați coloana la valoarea NOW():

  ```
  INSERT INTO nume_tabel (dt_col, ...) VALUES(NOW(), ...)
  ```

La fiecare actualizare ulterioară a înregistrării, lăsați coloana nemodificată:

```
UPDATE nume_tabel SET /* orice IN AFARA DE coloana dt_col */ WHERE ...
```

Dacă doriți să folosiți coloanele TIMESTAMP pentru a păstra atât o valoare a orei creării, cât și a orei ultimei modificări, puteți face aceasta folosind o coloană TIMESTAMP pentru valoarea orei modificării, respectiv o altă coloană TIMESTAMP pentru valoarea orei creării. Verificați ca ora modificării să fie inclusă în prima coloană TIMESTAMP, astfel încât să fie configurată când înregistrarea este creată sau modificată. Faceți din coloana cu ora creării cea de-a doua coloană TIMESTAMP și inițializați-o la valoarea NOW() atunci când creați înregistrări noi. Astfel, valoarea sa va reflecta momentul creării înregistrării și nu se va modifica ulterior.

Tipul de coloană YEAR

YEAR este un tip de coloană pe un octet folosit pentru reprezentarea eficientă a valorilor anilor. Domeniul de valori este cuprins între 1901 și 2155. Puteți folosi tipul YEAR atunci când doriți să stocați informații despre date calendaristice, dar aveți nevoie numai de an, cum ar fi anul nașterii, anul alegerii în funcția de președinte și altele. Când nu aveți nevoie de o valoare de tip dată completă, YEAR este mult mai eficient din punctul de vedere al spațiului utilizat decât alte tipuri de date.

Declarația unei coloane YEAR poate include o specificare a lățimii de afișare M, care poate fi 4 sau 2. Dacă M este omis dintr-o declarație de tip YEAR, valoarea prestabilită este 4.

TINYINT are aceeași dimensiune a spațiului de stocare ca și YEAR (un octet), dar nu și același domeniu. Pentru a acoperi aceeași perioadă de timp ca și YEAR folosind un tip întreg, aveți nevoie de un SMALLINT, care necesită o cantitate dublă de spațiu. Dacă intervalul de timp în ani pe care trebuie să-l reprezentați coincide cu domeniul tipului YEAR, cel din urmă folosește spațiul într-un mod mai eficient decât SMALLINT. Un alt avantaj al coloanei YEAR față de o coloană cu valori întregi este acela că MySQL va converti valorile din două cifre în valori cu patru cifre, folosind regulile obișnuite de „ghicire" a anului din MySQL. De exemplu, 97 și 14 devin 1997, respectiv 2014. Totuși, rețineți că inserția valorii numerice 00 va avea ca rezultat stocarea valorii 0000, nu a valorii 2000. Dacă doriți ca o valoare zero să fie convertită la 2000, trebuie să o specificați sub formă de șir: "00".

Atributele coloanelor de tip dată și oră

Nu există nici un atribut specific coloanelor de tip dată și oră. Atributele generale NULL sau NOT NULL pot fi specificate pentru oricare dintre tipurile dată și oră. Dacă nu specificați nici unul din aceste atribute, NULL este cel prestabilit. De asemenea, puteți specifica o valoare prestabilită folosind atributul DEFAULT. Dacă nu specificați o valoare prestabilită, o asemenea valoare va fi aleasă automat. Valoarea prestabilită este NULL pentru coloanele care pot conține NULL. În caz contrar, valoarea prestabilită pentru tipul respectiv este „zero".

Lucrul cu coloane de tip dată și oră

MySQL încearcă să interpreteze valorile de tip dată și oră într-o varietate de formate. Valorile DATE pot fi specificate în oricare din următoarele formate, inclusiv formele șir și numerice. Tabelul 2.13 prezintă formatele permise pentru fiecare dintre tipurile dată și oră.

Tabelul 2.13 Formate de intrare pentru coloanele de tip dată și oră

Tip	Formate permise
DATETIME, TIMESTAMP	"AAAA-LL-ZZ hh:mm:ss"
	"AA-LL-ZZ hh:mm:ss"
	"AAAALLZZhhmmss"
	"AALLZZhhmmss"
	AAAALLZZhhmmss
	AALLZZhhmmss
DATE	"AAAA-LL-ZZ"
	"AA-LL-ZZ"
	"AAAALLZZ"
	"AALLZZ"
	AAAALLZZ
	AALLZZ
TIME	"hh:mm:ss"
	"hhmmss"
	hhmmss
YEAR	"AA"
	AAAA
	AA

Formatele cu două cifre pentru valoarea anului sunt interpretate folosind regulile descrise în secțiunea „Interpretarea valorilor ambigui ale anilor". Pentru formate de tip șir care includ caractere de delimitare, nu trebuie să folosiți cratimele pentru date, respectiv caracterul două puncte pentru ore. Ca delimitator poate fi folosit orice semn de punctuație. Interpretarea valorilor depinde de context, nu de delimitator. De exemplu, deși orele sunt specificate de obicei folosind delimitatorul două puncte, MySQL nu va interpreta o valoare care conține acest caracter ca o oră într-un context unde se așteaptă o dată. În plus, pentru formatele de tip șir care includ delimitatori, nu trebuie să specificați două cifre pentru valorile lunilor, zilelor, orelor, minutelor sau secundelor care sunt mai mici decât 10. Următoarele formate sunt toate echivalente între ele:

```
"2012-02-03 05:04:09"
"2012-2-03 05:04:09"
"2012-2-3 05:04:09"
"2012-2-3 5:04:09"
"2012-2-3 5:4:09"
"2012-2-3 5:4:9"
```

Observați că valorile cu zerouri inițiale pot fi interpretate diferit, în funcție de faptul dacă sunt specificate ca șiruri sau ca numere. Șirul "001231" va fi văzut ca o valoare cu șase cifre și va fi interpretat sub forma "2000-12-31" pentru o coloană DATE, respectiv ca "2000-12-31 00:00:00" pentru o coloană DATETIME. Pe de altă parte, numărul 001231 va fi asimilat cu 1231 după ce a „trecut" prin analizor, moment după care interpretarea devine problematică. Aceasta este o situație unde cel mai bine este să furnizați o valoare de tip șir sau să folosiți o valoare complet determinată în cazul în care utilizați numere (adică 20001231 pentru DATE, respectiv 20001231000000 pentru DATETIME).

În general, puteți atribui la discreție valori între tipurile DATE, DATETIME și TIMESTAMP, deși trebuie să țineți cont de anumite restricții:

- Dacă repartizați o valoare DATETIME sau TIMESTAMP într-o coloană DATE, partea de oră este eliminată.

- Dacă repartizați o valoare DATE într-o coloană DATETIME sau TIMESTAMP, partea de oră a valorii rezultante primește valoarea zero.
- Tipurile respective au domenii diferite. În particular, TIMESTAMP are un domeniu mai limitat (între 1970 și 2037) deci, de exemplu, nu puteți repartiza o valoare DATETIME anterioară lui 1970 într-o coloană TIMESTAMP și să vă așteptați la rezultate rezonabile. De asemenea, într-o coloană TIMESTAMP nu puteți plasa valori din viitorul îndepărtat.

MySQL furnizează multe funcții pentru lucrul cu valorile de tip dată și oră. Pentru mai multe informații, vezi Anexa C.

Interpretarea valorilor ambigue ale anilor

Pentru toate coloanele de tip dată și oră care includ o parte care conține anul (DATE, DATETIME, TIMESTAMP, YEAR), MySQL manipulează valori care conțin ani exprimați prin două cifre convertind aceste valori în ani exprimați cu patru cifre. Această conversie se produce în conformitate cu următoarele reguli[7]:

- Valorile anilor cuprinse între 00 și 69 se transformă în valori din intervalul 2000-2069
- Valorile anilor cuprinse între 70 și 99 se transformă în valori din intervalul 1970-1999

Puteți vedea cel mai ușor efectul acestor reguli prin repartizarea a două valori cu două cifre diferite într-o coloană YEAR și apoi regăsind rezultatele. De asemenea, exemplul va scoate în evidență ceva ce merită reținut:

```
mysql> CREATE TABLE y_tabel (y YEAR);
mysql> INSERT INTO y_tabel VALUES(68),(69),(99), (00);
mysql> SELECT * FROM y_tabel;
```

y
2068
1969
1999
0000

Observați că 00 a fost convertit în 0000, nu în 2000. Aceasta deoarece 0 este o valoare absolut corectă pentru tipul YEAR; dacă inserați un zero numeric, atunci veți obține un zero numeric. Pentru a obține 2000, inserați șirul "0" sau "00". Vă puteți asigura că MySQL „vede“ un șir și nu un număr prin inserția valorilor YEAR folosind CONCAT(). Această funcție returnează întotdeauna un rezultat șir, indiferent dacă argumentul său este un șir sau un număr.

În orice caz, rețineți că regulile pentru conversia anilor compuși din două cifre în ani compuși din patru cifre asigură numai intuirea în limite rezonabile a valorii. Nu există nici o modalitate ca MySQL să fie sigur de semnificația unui an exprimat prin două cifre atunci când nu se specifică secolul. Dacă regulile de conversie din MySQL nu generează valorile dorite, soluția este evidentă: furnizați date fără echivoc, cu ani exprimați prin patru cifre.

[7] În MySQL 4.0, regulile vor suferi o oarecare schimbare, prin aceea că 69 va fi convertit în 1969, nu în 2069, în conformitate cu regulile specificate prin standardul UNIX X/Open. – N.A.

Alegerea tipurilor de coloană

Secțiunea anterioară descrie diferitele tipuri de coloane MySQL din care puteți alege, precum și proprietățile generale ale acestor tipuri, ca tipurile de valori pe care le pot conține, cantitatea de spațiu de stocare pe care o pot ocupa și altele. Dar cum alegeți tipurile de coloane pe care le veți folosi când creați un tabel? Această secțiune discută despre aspectele care trebuie avute în vedere pentru a vă ajuta să alegeți.

Cele mai „generale“ tipuri de coloane sunt tipurile șir. În aceste coloane puteți stoca orice, deoarece numerele și datele pot fi reprezentate sub formă de șir. Deci, de ce n-ați declara toate coloanele sub formă de șiruri și să terminați povestea? Să ne gândim la un singur exemplu. Să presupunem că aveți valori care arată ca niște numere. Le puteți reprezenta ca numere, dar chiar trebuie să o faceți? Ce se întâmplă dacă procedați astfel?

În primul rând, probabil că veți folosi o cantitate mai mare de spațiu, deoarece numerele pot fi stocate într-un mod mult mai eficient decât șirurile. De asemenea, veți remarca unele diferențe între rezultatele interogărilor, datorită modalităților diferite de manipulare a șirurilor și a numerelor. De exemplu, ordinea de sortare pentru numere nu este aceeași ca pentru șiruri. Numărul 2 este mai mic decât numărul 11, dar șirul "2" este, din punct de vedere lexicografic, mai mare decât șirul "11". Puteți rezolva această problemă folosind coloana într-un context numeric, ca acesta:

```
SELECT nume_coloana + 0 AS num... ORDER BY num
```

MySQL are „problema anului 2000“?

MySQL nu are problema anului 2000, deoarece stochează datele intern sub formă de ani cu patru cifre, dar este răspunderea dumneavoastră să furnizați de la bun început date care să aibă ca rezultat stocarea valorilor adecvate. Adevărata problemă cu interpretarea anilor cu două cifre nu este dată de MySQL, ci de dorința umană de a „o lua pe scurtătură“ și de a furniza date echivoce. Dacă doriți să vă asumați riscul, sunteți liber să o faceți. Este riscul dumneavoastră, iar regulile de determinare a anului din MySQL sunt adecvate în multe situații. Este bine să știți, însă, că există situații când *trebuie* să introduceți ani din patru cifre. De exemplu, pentru a introduce datele de naștere și de deces în tabelul presedinte, care conține președinții americani începând din secolul al XVIII-lea, se impune utilizarea a patru cifre. Valorile din aceste coloane acoperă mai multe secole, deci a lăsa MySQL să „ghicească“ secolul pe baza unui an din două cifre este categoric o opțiune eronată.

Prin adăugarea unui zero la coloană se forțează o sortare numerică, dar este oare rezonabil? În general, probabil că nu. A determina MySQL să trateze coloana ca pe un număr, nu ca pe un șir, are câteva implicații importante. Se forțează o conversie șir-număr pentru fiecare valoare din coloană, ceea ce este ineficient. De asemenea, transformarea coloanei într-o coloană cu valori calculate împiedică MySQL să folosească un index pentru coloană, ceea ce reduce și mai mult viteza interogării. Nici una din aceste degradări ale performanței nu se va produce dacă stocați de la bun început valorile sub formă de numere. Simpla opțiune de a folosi o reprezentare în locul alteia are implicații pentru cerințele privind dimensiunea spațiului de stocare, manipularea interogărilor și performanțele de prelucrare.

Exemplul anterior ilustrează faptul că la alegerea tipurilor de coloane intră în joc mai mulți factori. Lista următoare parcurge rapid factorii la care trebuie să vă gândiți atunci când selectați un tip pentru o coloană.

- **Ce tip de valori va conține coloana?** Numere? Șiruri? Date calendaristice? Aceasta este o întrebare evidentă, dar trebuie să o puneți. Puteți reprezenta orice tip de valoare sub formă de șir, dar, așa cum am văzut anterior, probabil veți obține performanțe superioare dacă folosiți alte tipuri mai adecvate pentru valorile numerice. (Acest lucru este de asemenea valabil pentru valorile de tip dată și oră.) Totuși, evaluarea tipului de valori cu care lucrați nu este în mod obligatoriu ceva banal, mai ales dacă este vorba despre datele altcuiva. Este deosebit de important să întrebați care este tipul de valori pe care îl vor conține coloanele în cazul în care configurați un tabel pentru altcineva și trebuie să fiți sigur că puneți destule întrebări cu scopul de a primi suficiente informații pentru a lua o decizie inspirată.
- **Valorile dumneavoastră se găsesc într-un anumit domeniu particular?** Dacă sunt întregi, vor fi întotdeauna diferite de zero? Dacă da, puteți folosi `UNSIGNED`. Dacă sunt șiruri, vor fi întotdeauna selectate dintr-un set fixat de valori? Dacă da, veți găsi util tipul `ENUM` sau `SET`.

  Există un compromis între domeniul unui tip și cantitatea de spațiu de stocare pe care o folosește. Cât de „mare" este tipul de care aveți nevoie? Pentru numere, puteți alege tipuri mici, cu un domeniu limitat de valori, respectiv tipuri mari, care sunt, în esență, nelimitate. Pentru șiruri, le puteți face scurte sau lungi și nu veți alege declarația `CHAR(255)` dacă toate valorile pe care doriți să le stocați conțin mai puțin de 10 caractere.
- **Care sunt aspectele legate de performanță și eficiență?** Unele tipuri pot fi prelucrate mai eficient decât altele. Operațiile numerice, în general, pot fi efectuate mai rapid decât operațiile cu șiruri. Șirurile scurte pot fi comparate mai ușor decât șirurile lungi și implică, de asemenea, o suprasarcină mai redusă asupra discului. Performanțele sunt mai ridicate pentru tipurile de lungime fixă decât pentru cele de lungime variabilă.
- **Cum doriți să fie comparate valorile dumneavoastră?** Pentru șiruri, comparațiile pot sesiza sau nu diferența între majuscule și minuscule. Aici, opțiunile dumneavoastră mai pot afecta și sortarea, care este bazată pe comparații.
- **Intenționați să indexați o coloană?** Dacă da, acest lucru influențează opțiunea dumneavoastră privind tipul coloanei, deoarece unele versiuni de MySQL nu vă permit să indexați anumite tipuri, precum `BLOB` și `TEXT`. De asemenea, unele versiuni de MySQL impun ca o coloană indexată să fie declarată ca `NOT NULL`, ceea ce afectează posibilitatea dumneavoastră de a folosi valori `NULL`.

Acum, să luăm în considerare fiecare dintre aceste probleme în detaliu. Dar, înainte de a trece la fapte, permiteți-mi un punct de vedere: doriți să faceți cele mai bune alegeri posibile privind tipurile de coloane atunci când creați un tabel, dar, dacă faceți o alegere care nu se dovedește a fi optimă, nu este sfârșitul lumii. Puteți folosi `ALTER TABLE` pentru a înlocui tipul cu unul mai bun, ceea ce se poate reduce la înlocuirea unui `SMALLINT` cu un `MEDIUMINT` după ce ați descoperit că datele dumneavoastră conțin valori mai mari decât cele la care v-ați gândit inițial. Această operație poate fi, însă, complicată, cum ar fi înlocuirea unui `CHAR` cu un `ENUM` cu un set bine precizat de valori permise. În MySQL 3.23 și versiunile ulterioare, puteți folosi `PROCEDURE ANALYSE()` pentru a obține informații despre coloanele tabelului dumneavoastră, cum ar fi valorile minime și maxime, precum și un tip optim propus care să acopere domeniul de valori dintr-o coloană. Această funcție vă poate ajuta să concluzionați că se poate folosi un tip mai mic, care poate îmbunătăți performanțele interogărilor care implică tabelul și care reduce cantitatea de spațiu necesară pentru stocarea tabelului.

Ce tip de valori va conține coloana?

Primul lucru la care vă gândiți atunci când încercați să vă decideți asupra unui tip de coloană este genul de valori pentru care va fi folosită coloana, deoarece acesta este faptul cu implicațiile cele mai evidente pentru tipul pe care îl alegeți. În general, procedați într-un mod previzibil: stocați numerele în coloane numerice, șirurile în coloanele șir, respectiv datele și orele în coloane de tip dată și oră. Dacă numerele dumneavoastră au o parte fracționară, veți folosi un tip de coloană cu virgulă mobilă în locul unui tip întreg și așa mai departe. Uneori există și excepții. Aici, principiul este că trebuie să înțelegeți natura datelor dumneavoastră, pentru a fi capabil să alegeți tipul de coloană într-o manieră „informată". Dacă vă veți stoca propriile date, probabil că știți bine cum să le caracterizați. Pe de altă parte, dacă alte persoane vă solicită să le configurați un tabel, uneori datele problemei se schimbă. S-ar putea să nu aflați prea ușor care sunt elementele pe care le folosiți. Nu uitați să puneți suficiente întrebări pentru a afla tipurile de valori pe care trebuie să le conțină tabelul.

Să presupunem că vi se spune că o coloană trebuie să stocheze „cantitatea de precipitații". Este vorba de un număr? Sau este „în cea mai mare parte" numeric – adică de regulă, dar nu întotdeauna, este codificată sub forma unui număr? De exemplu, când urmăriți știrile la televizor, buletinul meteorologic include, în general, o măsură a precipitațiilor. Uneori este vorba de un număr (de exemplu „6 mm de ploaie"), dar alteori este o „urmă" de precipitații, ceea ce înseamnă „foarte puțin". Toate bune și frumoase în ceea ce privește buletinul meteorologic, dar ce înseamnă asta pentru stocarea într-o bază de date? Fie trebuie să cuantificați noțiunea de „urmă" sub forma unui număr, astfel încât să puteți folosi o coloană numerică pentru a înregistra cantitățile de precipitații, fie trebuie să folosiți un șir, astfel încât să puteți înregistra cuvântul „urmă". Alternativ, puteți veni cu un aranjament mai complicat, folosind o coloană numerică și o coloană șir, caz în care completați o coloană și inserați în cealaltă valoarea `NULL`. Este evident că doriți să evitați acea opțiune, dacă este posibil; tabelul ar deveni mai dificil de înțeles și îngreunează semnificativ scrierea interogărilor.

Probabil că eu aș încerca să stochez toate rândurile în format numeric și apoi să le convertesc conform necesităților de afișare. De exemplu, dacă orice cantitate de precipitații mai mică de 0,25 mm, dar diferită de zero, este considerată o cantitate de tip „urmă", puteți afișa valori din coloană astfel:

```
SELECT IF(precip>0 AND precip<.25,"urma",precip) FROM ...
```

Pentru calcule monetare, lucrați cu valori care conțin dolari și cenți[8]. Acestea par a fi valori cu virgulă mobilă, dar `FLOAT` și `DOUBLE` sunt supuse la erori prin rotunjire și pot fi inadecvate, cu excepția înregistrărilor în care aveți nevoie numai de o precizie aproximativă. Deoarece oamenii sunt destul de sensibili când este vorba de banii lor, probabil că veți avea nevoie de un tip de coloane care să permită o precizie perfectă. Aveți la dispoziție câteva opțiuni:

- Puteți reprezenta sumele de bani ca un tip `DECIMAL(M, 2)`, alegând `M` ca lățime maximă adecvată pentru domeniul de valori de care aveți nevoie. Astfel, obțineți valori cu virgulă mobilă cu precizie de două cifre după virgulă. Avantajul tipului `DECIMAL` este acela că valorile sunt reprezentate sub formă de șiruri și nu sunt supuse erorilor prin rotunjire. Dezavantajul este că operațiile cu șiruri sunt mai puțin eficiente decât cele cu valori reprezentate intern sub formă de numere.
- Puteți reprezenta intern toate valorile monetare sub formă de cenți, folosind un tip întreg. Avantajul este că toate calculele sunt efectuate intern folosind întregi, ceea ce determină o mare viteză de calcul. Dezavantajul este că va trebui să convertiți valorile la intrare sau ieșire, prin înmulțire sau împărțire la 100.

Unele valori sunt în mod evident numerice, dar trebuie să determinați dacă veți folosi un tip întreg sau cu virgulă mobilă. Trebuie să întrebați care sunt unitățile de măsură și precizia cerută. Precizia dată de unitățile întregi este suficientă sau trebuie să reprezentați și unități fracționare? Această întrebare vă poate ajuta să faceți diferența dintre tipurile de coloană întregi și cele cu virgulă mobilă. De exemplu, dacă reprezentați greutăți, puteți folosi o coloană cu valori întregi dacă înregistrați valorile până la cea mai apropiată cantitate exprimată în kilograme. Veți folosi o coloană cu virgulă mobilă dacă doriți să înregistrați unități fracționare. În unele situații, puteți folosi chiar și câmpuri multiple, de exemplu dacă doriți să înregistrați greutatea în kilograme și grame.

Înălțimea este un alt tip numeric de informații pentru care există numeroase posibilități de reprezentare:

- **Un șir de genul `"6-2"` pentru o valoare de genul „6 picioare și 2 inch[9]".** Această opțiune are avantajul de a avea o formă simplu de examinat și de înțeles (categoric mai simplă decât „74 inch"), dar această categorie de valori este mai dificil de utilizat pentru operații matematice, precum însumarea sau calculul mediei.
- **Un câmp numeric pentru valorile exprimate în picioare și altul pentru valorile exprimate în inch.** Aceasta este o formă ceva mai ușor de întrebuințat pentru operațiile matematice, dar două câmpuri sunt mai dificil de folosit decât unul singur.
- **Un câmp numeric reprezentând valorile în inch.** Este cel mai ușor de utilizat de către o bază de date, dar cel mai greu de înțeles pentru oameni. Dar nu uitați că nu trebuie să prezentați valorile în formatul pe care îl folosiți pentru a lucra cu ele. Puteți reformata valorile pentru o afișare mai semnificativă folosind numeroasele funcții ale sistemului MySQL. Aceasta înseamnă că opțiunea de față este cea mai bună modalitate de reprezentare a înălțimii.

[8] Am preferat să păstrăm textul original, deoarece „marea" noastră monedă națională a ajuns atât de mică, încât vorbind despre lei și bani (adică a suta parte dintr-un leu) în momentul actual riscăm să nu producem decât râsul (amar sau ironic) al cititorului român... – N.T.

[9] În sistemul metric anglo-saxon, unități de măsură a lungimii. 1 picior = 33 cm, 1 inch = 2,54 cm. – N.T.

Dacă trebuie să stocați informații despre date calendaristice, valorile respective includ și ora? Cu alte cuvinte, va fi *vreodată* nevoie să includă și ora? MySQL nu furnizează un tip de date care să aibă o parte opțională pentru oră: `DATE` nu are niciodată o oră, iar `DATETIME` trebuie să aibă o oră. Dacă ora este într-adevăr opțională, folosiți o coloană `DATE` pentru a înregistra data, respectiv o coloană `TIME` separată pentru a include ora. Apoi permiteți coloanei `TIME` să ia valoarea `NULL` și interpretați această valoare ca „fără oră":

```
CREATE TABEL tabelul_meu
(
    data DATE NOT NULL,
    ora TIME NULL
)
```

Un tip de situație când este deosebit de important să determinați dacă aveți sau nu nevoie de o valoare a orei se produce atunci când uniți două tabele cu o relație de tip master-detaliu, tabele care sunt „legate" în funcție de informațiile legate de dată.

Să presupunem că sunteți coordonatorul unei activități de cercetare care implică subiecți ce intră în biroul dumneavoastră pentru a fi testați. După un set inițial standard de teste, puteți efectua mai multe teste suplimentare în aceeași zi, unde opțiunea testelor variază în funcție de rezultatele testelor inițiale. Puteți reprezenta aceste informații folosind o relație master-detaliu, în care informațiile de identificare a subiectului și testele standard inițiale sunt stocate într-o înregistrare master, iar toate testele suplimentare sunt stocate sub formă de rânduri într-un tabel secundar cu detalii. Apoi, corelați cele două tabele în funcție de identificatorul subiectului și data la care au avut loc testele.

Întrebarea la care trebuie să răspundeți în această situație este dacă puteți folosi numai data sau dacă vă trebuie atât data, cât și ora. Acest lucru depinde de faptul dacă un subiect poate parcurge procedura de testare de mai multe ori, pe durata aceleiași zile. Dacă da, înregistrați ora (de exemplu ora la care începe procedura) folosind fie o coloană `DATETIME`, fie coloane `DATE` și `TIME` separate, care trebuie completate amândouă. Fără valoarea orei, nu veți putea asocia înregistrările detaliu ale unui subiect cu înregistrările master adecvate dacă subiectul a fost testat de două ori într-o zi.

Am auzit oameni spunând: „N-am nevoie de oră; nu voi testa niciodată un subiect de două ori în aceeași zi." Uneori au dreptate, dar i-am văzut pe o parte din aceiași oameni ajungând ulterior să se întrebe cum pot preveni asocierea înregistrărilor detaliu cu o înregistrare master greșită, după ce au introdus date pentru subiecți care au fost testați de mai multe ori într-o zi. Scuze, dar atunci e prea târziu!

Uneori puteți rezolva această problemă prin inserția *post factum* în tabele a unei coloane `TIME`. Din păcate, înregistrările existente sunt dificil de remediat, cu excepția situațiilor când aveți o sursă independentă de date, cum sunt înregistrările originale scrise. Altfel, nu aveți nici o modalitate de a „clarifica misterul" înregistrărilor detaliu ambigui, pentru a le asocia cu înregistrarea master adecvată. Chiar dacă aveți o sursă independentă de informații, aceasta este o operație încurcată și poate cauza probleme aplicațiilor pe care le-ați scris deja cu scopul de a utiliza tabelele. Cel mai bine este să explicați problema

posesorilor tabelelor și să vă asigurați că dispuneți de o bună caracterizare a valorilor datelor înainte de a le crea tabelele.

Uneori puteți avea date incomplete, lucru ce poate influența opțiunea dumneavoastră privind tipurile de coloane. De exemplu, adunați date de naștere și deces pentru cercetări de natură genealogică, iar uneori tot ce puteți afla este anul în care cineva s-a născut sau a decedat, nu și data exactă. Dacă folosiți o coloană DATE, nu puteți introduce o dată decât dacă dispuneți de toate elementele acesteia. Dacă doriți să puteți înregistra informațiile pe care le aveți, oricare ar fi acestea, chiar dacă sunt incomplete, s-ar putea să fiți obligat să folosiți câmpuri separate pentru an, lună și zi. Apoi puteți introduce părțile de informații pe care le aveți și inserați valoarea NULL pentru celelalte informații. În MySQL 3.23 și versiunile ulterioare există și o altă posibilitate, care permite părților care conțin ziua, respectiv luna și data, să ia valoarea 0. Asemenea date „neclare" pot fi folosite pentru a reprezenta valori de date incomplete.

Valorile dumneavoastră se încadrează într-un anumit domeniu?

Dacă v-ați oprit asupra unei categorii generale din care să selectați un tip pentru o coloană, stabilirea domeniului de valori pe care doriți să-l reprezentați vă va ajuta să reduceți plaja de opțiuni la un anumit tip din cadrul acelei categorii. Să presupunem că doriți să stocați valori întregi. Domeniul valorilor dumneavoastră determină tipurile pe care le puteți folosi. Dacă aveți nevoie de valori în domeniul 0-1000, puteți folosi orice tip cuprins între SMALLINT și BIGINT. Dacă valorile dumneavoastră ajung până la două milioane, nu puteți folosi SMALLINT, iar opțiunile dumneavoastră variază de la MEDIUMINT la BIGINT. Apoi, trebuie să selectați un tip dintre posibilitățile existente.

Puteți, desigur, să folosiți pur și simplu cel mai mare tip pentru categoria de valoare pe care doriți să o stocați (BIGINT pentru exemplele din paragraful anterior). În general, totuși, trebuie să folosiți cel mai mic tip care este suficient de mare pentru intențiile dumneavoastră. Procedând astfel, veți reduce la minimum cantitatea de spațiu folosită de tabelele dumneavoastră, iar acestea vă vor oferi performanțe superioare, deoarece coloanele mai mici pot fi, de obicei, prelucrate mai rapid decât coloanele mai mari.

Dacă nu cunoașteți domeniul de valori pe care va trebui să-l puteți reprezenta, trebuie să vă folosiți intuiția sau să utilizați BIGINT pentru a lua în calcul cel mai rău caz cu putință. (Rețineți că, dacă vă dați cu părerea și folosiți un tip care se dovedește a fi prea mic, nu este totul pierdut; puteți folosi mai târziu ALTER TABLE pentru a face coloana mai „încăpătoare".)

În capitolul 1, am creat un tabel puncte pentru proiectul de evidență a rezultatelor școlare, care conținea o coloană puncte pentru înregistrarea punctajelor obținute la chestionare și la teste. Tabelul fusese creat folosind INT pentru a păstra simplitatea expunerii, dar acum puteți vedea că, dacă punctajele se află în domeniul 0-100, o opțiune mai bună ar fi TINYINT UNSIGNED, deoarece ar folosi un spațiu de stocare mai redus.

Domeniul de valori din datele dumneavoastră influențează de asemenea atributele pe care le puteți folosi pentru tipul coloanei dumneavoastră. Dacă valorile nu sunt niciodată negative, puteți folosi UNSIGNED; în caz contrar, nu puteți.

Tipurile șir nu au un „domeniu" în sensul domeniului coloanelor numerice, dar ele au o lungime, iar lungimea maximă de care aveți nevoie afectează tipurile de coloane pe care le puteți folosi. Dacă șirurile dumneavoastră sunt mai scurte de 256 caractere, puteți folosi CHAR, VARCHAR, TINYTEXT sau TINYBLOB. Dacă aveți nevoie de șiruri mai lungi, puteți folosi un tip TEXT sau BLOB, dar asta înseamnă că CHAR și VARCHAR nu mai constituie opțiuni viabile pentru dumneavoastră.

Pentru coloanele șir pe care le veți folosi pentru a reprezenta un set fix de valori, puteți lua în considerare utilizarea unui tip de coloane ENUM sau SET. Acestea pot constitui opțiuni utile, deoarece sunt reprezentate intern sub formă de numere. Operațiile cu aceste tipuri sunt efectuate numeric, deci sunt mai eficiente decât alte tipuri de șiruri. De asemenea, pot fi mai compacte decât alte tipuri de coloane șir, ceea ce determină economii de spațiu.

Când caracterizați domeniul de valori cu care trebuie să lucrați, termenii cei mai inspirați sunt „întotdeauna" și „niciodată" (de exemplu „întotdeauna mai mic decât 1000" sau „niciodată negativ"), deoarece vă ajută să reduceți mai mult opțiunile privind tipurile de coloane. Dar păziți-vă să folosiți acești termeni atunci când nu sunt pe deplin justificați. Fiți atent mai ales când vă consultați cu alte persoane cu privire la datele lor și respectivii încep să vă bombardeze cu acești doi termeni. Când omul zice „întotdeauna" sau „niciodată", asigurați-vă că vorbește serios. Uneori oamenii spun că datele lor au întotdeauna o anumită caracteristică, pe când în realitate vor să spună „aproape întotdeauna".

De exemplu, să presupunem că proiectați un tabel pentru anumite persoane și că acestea vă spun: „Rezultatele la test sunt întotdeauna cuprinse între 0 și 100." Bazându-vă pe această afirmație, alegeți tipul TINYINT și îi dați atributul UNSIGNED, deoarece valorile sunt întotdeauna diferite de zero. Apoi, descoperiți că persoanele care codifică datele pentru introducerea în baza de date folosesc uneori –1 pentru a semnala că un elev a lipsit pe motiv de boală. Măi să fie! P-asta nu v-au spus-o. Poate fi acceptabil să folosiți NULL pentru a reprezenta asemenea valori dar, dacă nu se poate, va trebui să înregistrați un –1, după care nu mai puteți folosi o coloană UNSIGNED. (Pas alergător, direcția ALTER TABLE!)

Uneori, deciziile despre aceste tipuri de cazuri se pot lua mai ușor punând o întrebare simplă: Există *vreodată* excepții? Dacă se produce *vreodată* un caz excepțional, chiar și o singură dată, trebuie să fiți pregătit. Veți descoperi că persoanele care discută cu dumneavoastră despre proiectarea unei baze de date sunt invariabil de părere că, dacă excepțiile nu se produc foarte frecvent, atunci nu contează. Când creați un tabel, nu puteți gândi astfel. Întrebarea pe care trebuie să o puneți nu este *cât* de frecvent se produc excepții, ci *dacă* acestea se produc. Dacă excepțiile se produc, trebuie să țineți cont de ele.

Care sunt problemele de performanță și de eficiență?

Opțiunea dumneavoastră privind tipul de coloane poate influența performanțele interogărilor în numeroase moduri. Dacă aveți în vedere îndrumările generale discutate în secțiunile anterioare, veți putea alege tipurile care vor contribui la prelucrarea mai eficientă în MySQL a datelor dumneavoastră.

Operații numerice sau cu șiruri

Operațiile numerice sunt, în general, mai rapide decât operațiile cu șiruri. Să ne gândim la operațiile de comparație. Numerele pot fi comparate dintr-o singură operație. Comparațiile între șiruri pot necesita mai multe comparații între octeți, care sunt cu atât mai multe cu cât șirurile devin mai lungi.

Dacă o coloană șir are un număr limitat de valori, folosiți un tip `ENUM` sau `TYPE`, pentru a beneficia de avantajele operațiilor cu numere. Aceste tipuri sunt reprezentate intern sub formă de numere și pot fi prelucrate mai eficient.

Să ne gândim la reprezentările alternative pentru șiruri. Uneori, puteți îmbunătăți performanțele prin reprezentarea valorilor șir sub formă de numere. De exemplu, pentru a reprezenta numerele IP în notația cu puncte, cum este `192.168.0.4`, ați putea folosi un șir. Dar, ca alternativă, puteți converti numerele IP în întregi prin stocarea fiecărei părți a formei cu puncte într-un octet din cei patru ai unui tip `INT UNSIGNED`. Astfel, se va economisi spațiu și va crește viteza de căutare. Pe de altă parte, reprezentarea numerelor IP sub formă de valori `INT` ar face dificilă stabilirea de corespondențe cu un model, cum sunt cele pe care le-ați face dacă doriți să căutați numere dintr-o subrețea dată. Deci, nu puteți ține cont numai de aspectele legate de spațiu; trebuie să decideți reprezentarea cea mai adecvată în funcție de modul în care doriți să utilizați valorile.

Tipuri mai mari sau mai mici

Tipurile mai mici pot fi prelucrate mai rapid decât tipurile mai mari. Unul din motive ar fi acela că ocupă spațiu mai puțin și implică o suprasarcină mai redusă pentru activitatea discului. Pentru șiruri, timpul de prelucrare se află într-o relație directă cu lungimea șirului.

În general, tabelele mai mici sunt mai rapide, deoarece prelucrarea interogărilor necesită mai puține operații de intrare-ieșire cu discul. Pentru coloane care folosesc tipuri de dimensiune fixă, alegeți cel mai mic tip care va conține domeniul impus de valori. De exemplu, nu folosiți `BIGINT` dacă se poate utiliza `MEDIUMINT`. Nu folosiți `DOUBLE` dacă aveți nevoie numai de o precizie `FLOAT`. Puteți economisi spațiu chiar și în cazul tipurilor de dimensiune variabilă. Un `BLOB` folosește 2 octeți pentru a înregistra lungimea valorii, un `LONGBLOB` folosește 4 octeți. Dacă stocați valori care nu depășesc niciodată 64OKB, utilizarea tipului `BLOB` vă economisește 2 octeți pentru fiecare valoare. (Considerații similare sunt valabile și pentru tipurile `TEXT`, desigur.)

Tipuri de lungime fixă sau variabilă

În general, tipurile de lungime fixă pot fi prelucrate mai rapid decât tipurile de lungime variabilă:

- În cazul coloanelor de lungime variabilă, un tabel în care se efectuează numeroase actualizări sau ștergeri va prezenta un grad superior de fragmentare, datorită dimensiunilor diferite a înregistrărilor. Va trebui să rulați periodic procedura `OPTIMIZE TABLE` pentru conservarea performanțelor. Aceasta nu constituie o problemă în cazul rândurilor de lungime fixă.
- Tabelele cu rânduri de lungime fixă sunt mai ușor de reconstruit în cazul „căderii" unui tabel, deoarece poate fi determinat începutul fiecărei înregistrări. Acest lucru nu este valabil la înregistrările de lungime variabilă și nu constituie o problemă de performanță în ceea ce privește prelucrările interogărilor, dar poate accelera în mod evident procesul de refacere a tabelului.

Dacă în tabelul dumneavoastră există coloane cu lungime variabilă, conversia lor la coloane cu lungime fixă va determina o îmbunătățire a performanțelor, deoarece înregistrările cu lungime fixă sunt mai ușor de prelucrat. Înainte de a încerca această operație, țineți cont de următoarele aspecte:

- Utilizarea coloanelor cu lungime fixă implică un compromis. Aceste coloane sunt mai rapide, dar ocupă o cantitate mai mare de spațiu. Coloanele `CHAR(n)` ocupă întotdeauna n octeți pentru fiecare valoare (chiar și pentru valorile vide), deoarece valorile sunt completate cu spații finale atunci când sunt stocate în tabel. Coloanele `VARCHAR(n)` ocupă mai puțin spațiu, deoarece este alocat numai spațiul necesar pentru stocarea fiecărei valori, plus un octet în fiecare valoare pentru înregistrarea lungimii. Astfel, dacă alegeți între coloanele `CHAR` și `VARCHAR`, se face un compromis între timp și spațiu. Dacă viteza este problema dumneavoastră principală, folosiți coloanele `CHAR` pentru a obține avantajele de performanță ale coloanelor de lungime fixă. Dacă spațiul este la mare preț, folosiți coloanele `VARCHAR`.
- Nu puteți converti o singură coloană de lungime variabilă; trebuie să le convertiți pe toate. De asemenea, trebuie să convertiți simultan toate coloanele de lungime variabilă folosind o singură instrucțiune `ALTER TABLE`; în caz contrar, demersul dumneavoastră nu va avea nici un efect.
- Uneori, nu puteți folosi un tip de coloană cu lungime fixă, chiar dacă doriți acest lucru. Nu există tipuri de lungime fixă pentru șiruri mai lungi de 255 caractere, de exemplu.

Tipuri indexabile

Indexurile măresc mult viteza interogărilor, deci alegeți tipuri pe care le puteți indexa. Vezi secțiunea „Aspecte legate de indexare" pentru mai multe informații.

Tipuri NULL sau NOT NULL

Dacă declarați o coloană ca fiind `NOT NULL`, aceasta poate fi manipulată mai rapid, deoarece MySQL nu trebuie să verifice valorile coloanelor în timpul prelucrării interogărilor, pentru a vedea dacă acestea conțin sau nu valoarea `NULL`. De asemenea, se economisește un bit pentru fiecare rând din tabel.

Evitarea valorii `NULL` în coloane poate simplifica interogările, deoarece nu trebuie să vă mai gândiți la `NULL` ca la un caz special. În general, interogările mai simple sunt prelucrate mai rapid.

Îndrumările legate de performanță prezentate mai sus intră uneori în conflict. De exemplu, un rând cu lungime fixă care conține coloane `CHAR` va fi mai rapid decât un rând cu lungime variabilă care conține coloane `VARCHAR`, din punctul de vedere al capacității sistemului MySQL de localizare a rândurilor. Pe de altă parte, un asemenea rând va ocupa mai mult spațiu, deci implică o activitate suplimentară a discului. Din acest punct de vedere, `VARCHAR` poate fi mai rapid. Ca regulă empirică, puteți presupune că rândurile cu lungime fixă vor determina o creștere a performanțelor, chiar dacă se folosește o cantitate mai mare de spațiu. Pentru o aplicație deosebit de importantă, puteți implementa

un tabel în ambele moduri și eventual efectuați teste pentru a determina alternativa cea mai bună pentru aplicația respectivă.

Cum doriți să fie comparate valorile dumneavoastră?

Puteți determina compararea și sortarea tipurilor șir într-un mod sensibil sau nu la diferența între majuscule și minuscule prin modul în care declarați aceste tipuri. Tabelul 2.14 prezintă fiecare tip care nu este sensibil la diferența între majuscule și minuscule, precum și echivalentul său sensibil la o atare diferență. Unele tipuri (CHAR, VARCHAR) sunt binare sau nu în funcție de prezența sau absența cuvântului cheie BINARY în declarația coloanei. Caracterul binar al altor tipuri (BLOB, TEXT) este conținut implicit în numele tipului.

Tabelul 2.14 Sensibilitatea tipurilor șir la diferența între majuscule și minuscule

Tip non-binar (insensibil la diferența între majuscule și minuscule)	Tip binar (sensibil la diferența între majuscule și minuscule)
CHAR(*M*)	CHAR(*M*) BINARY
VARCHAR(*M*)	VARCHAR(*M*) BINARY
TINYTEXT	TINYBLOB
TEXT	BLOB
MEDIUMTEXT	MEDIUMBLOB
LONGTEXT	LONGBLOB

Rețineți că tipurile binare (sensibile la diferența între majuscule și minuscule) diferă de omoloagele lor non-binare (insensibile la diferența între majuscule și minuscule) numai în ceea ce privește comportarea în materie de comparare și sortare. Orice tip șir poate conține orice categorie de date. În particular, tipurile TEXT pot conține date binare fără nici un fel de probleme, în ciuda mențiunii „TEXT" din numele tipurilor de coloane.

Dacă doriți să folosiți o coloană atât pentru comparații sensibile la diferența între majuscule și minuscule, cât și pentru comparații insensibile la o asemenea diferență, folosiți un tip non-binar. Apoi, de fiecare dată când doriți o comparație sensibilă la diferența între majuscule și minuscule, folosiți cuvântul cheie BINARY pentru a forța tratarea unui șir ca pe o valoare de tip șir binar. De exemplu, în cazul în care coloana_mea este o coloană CHAR, o puteți compara în moduri diferite:

coloana_mea = "ABC"	Insensibil la diferența între majuscule și minuscule
BINARY coloana_mea = "ABC"	Sensibil la diferența între majuscule și minuscule
coloana_mea = BINARY "ABC"	Sensibil la diferența între majuscule și minuscule

Dacă aveți valori șir pe care doriți să le sortați într-o anumită ordine ne-lexicografică, aveți în vedere utilizarea unei coloane ENUM. Sortarea valorilor ENUM se produce în conformitate cu ordinea în care menționați valorile din enumerare în declarația coloanei, deci puteți determina sortarea acestor valori în orice ordine doriți.

Intenționați să indexați o coloană?

Indexurile permit sistemului MySQL să prelucreze interogările mai rapid. Alegerea indexurilor este o problemă discutată mai detaliat în capitolul 4, dar un principiu general precizează că acele coloane pe care le folosiți de obicei în clauzele WHERE pentru selectarea rândurilor constituie cele mai bune opțiuni pentru indexare.

Dacă doriți să indexați o coloană sau să o includeți într-un index cu mai multe coloane, există restricții privind tipurile pe care le puteți alege. În versiunile MySQL anterioare versiunii 3.23.2, coloanele indexate trebuie declarate NOT NULL, după cum tipurile BLOB și TEXT nu pot fi indexate. Aceste restricții au fost eliminate în MySQL 3.23.2, dar, dacă folosiți o versiune anterioară și nu puteți sau nu doriți să treceți la o versiune mai recentă, trebuie să lucrați cu aceste restricții. Totuși, puteți ocoli restricțiile în următoarele situații:

- Dacă puteți specifica o anumită valoare ca fiind specială, o puteți trata ca și cum ar avea aceeași semnificație ca și NULL. Pentru o coloană DATE, puteți desemna valoarea "0000-00-00" ca având semnificația „fără dată". Într-o coloană de tip șir, puteți preciza că un șir vid are semnificația „valoare lipsă". Într-o coloană numerică, puteți folosi –1 dacă, în mod normal, coloana va conține numai valori non-negative.
- Nu puteți indexa tipurile BLOB sau TEXT, dar, dacă șirurile dumneavoastră nu depășesc 255 de caractere, folosiți tipul de coloană echivalent VARCHAR și indexați-l pe acela. Puteți folosi VARCHAR(255) BINARY pentru valorile BLOB, respectiv VARCHAR(255) pentru valorile TEXT.

Intercorelarea problemelor legate de selectarea tipului de coloană

Nu puteți lua întotdeauna în considerare problemele legate de selectarea tipurilor de coloane ca și cum acestea ar fi independente una de alta. De exemplu, în cazul tipurilor numerice domeniul este corelat cu dimensiunea spațiului de stocare; când domeniul crește, aveți nevoie de o cantitate mai mare de spațiu, ceea ce afectează performanțele. Ca alt exemplu, luați în considerare implicațiile opțiunii de utilizare a atributului AUTO_INCREMENT pentru crearea unei coloane care să conțină numere de secvență unice. Opțiunea respectivă are numeroase consecințe, care implică tipul coloanei, indexarea și utilizarea valorii NULL:

- AUTO_INCREMENT este un atribut de coloană care trebuie folosit numai cu tipuri întregi. Astfel, opțiunile dumneavoastră sunt imediat limitate la tipurile cuprinse între TINYINT și BIGINT.
- Coloanele AUTO_INCREMENT trebuie indexate de așa manieră încât numărul curent maxim din cadrul secvenței să poată fi determinat rapid, fără a necesita o parcurgere completă a tabelului. În continuare, pentru a preveni reutilizarea numerelor din secvență, indexul trebuie să fie unic. Aceasta înseamnă că trebuie să declarați coloana ca PRIMARY KEY sau ca index de tip UNIQUE.
- Dacă versiunea dumneavoastră de MySQL este anterioară versiunii 3.23.2, coloanele indexate nu pot conține valori NULL, deci trebuie să declarați coloana ca fiind de tip NOT NULL.

Toate acestea înseamnă că nu puteți declara o coloană AUTO_INCREMENT astfel:

```
col tip_arbitrar AUTO_INCREMENT
```

O astfel de coloană se declară după cum urmează:

```
col tip_intreg AUTO_INCREMENT NOT NULL PRIMARY KEY
```

Sau astfel:

```
col tip_intreg AUTO_INCREMENT NOT NULL, UNIQUE(col)
```

O consecință suplimentară a utilizării atributului AUTO_INCREMENT este aceea că, atributul fiind destinat generării unei secvențe de valori pozitive, puteți la fel de bine să declarați o coloană AUTO_INCREMENT ca fiind UNSIGNED:

```
col tip_intreg UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY
col tip_intreg UNSIGNED AUTO_INCREMENT NOT NULL, UNIQUE(col)
```

Evaluarea expresiilor și conversia de tip

MySQL vă permite să scrieți expresii care includ constante, apeluri la funcții și referințe la coloane dintr-un tabel. Aceste valori pot fi combinate folosind diferite categorii de operatori, precum operatorii aritmetici sau de comparație, iar termenii unei expresii pot fi grupați cu ajutorul parantezelor.

Expresiile apar cel mai frecvent în lista de selecție a coloanei și în clauza WHERE a instrucțiunilor SELECT:

```
SELECT
    CONCAT(nume, ", ", prenume),
    (TO_DAYS(data_deces) - TO_DAYS(data_nastere) / 365)
FROM presedinte
WHERE
    data_nastere > "1900-1-1" AND data_deces IS NOT NULL
```

Fiecare coloană selectată reprezintă o expresie, ca și conținutul clauzei WHERE. De asemenea, expresiile mai apar în clauza WHERE a instrucțiunilor DELETE și UPDATE, în clauza VALUES() a instrucțiunilor INSERT și așa mai departe.

Când MySQL întâlnește o expresie, o evaluează pentru a calcula un rezultat. De exemplu (4*3)/(4-2) are rezultatul 6. Evaluarea expresiilor poate necesita conversii de tip. De exemplu, MySQL convertește numărul 960821 într-o dată, și anume "1996-08-21", atunci când numărul este folosit într-un context care necesită o valoare de tip dată.

Această secțiune discută modul în care puteți scrie expresii în MySQL, precum și care sunt regulile care guvernează diferitele categorii de conversii de tip pe care le execută MySQL în decursul procesului de evaluare a expresiilor. Fiecare dintre operatorii MySQL este prezentat aici, dar MySQL are un număr atât de mare de funcții, încât vom aborda numai câteva. Totuși, fiecare operator și fiecare funcție sunt prezentate în detaliu în Anexa C.

Scrierea expresiilor

O expresie se poate reduce la o singură constantă:

`0`	Constantă numerică
`"abc"`	Constantă de tip șir

Expresiile folosesc apeluri la funcții. Unele funcții iau argumente (valori plasate între paranteze), altele nu. Argumentele multiple trebuie să fie separate prin virgulă. Când invocați o funcție, argumentele pot fi delimitate prin spații, dar între numele funcției și paranteza de deschidere nu trebuie să existe nici un spațiu:

`NOW()`	Funcție fără argumente
`STRCMP("abc","def")`	Funcție cu două argumente
`STRCMP( "abc", "def" )`	Sunt permise spațiile în jurul argumentelor
`STRCMP ("abc","def")`	Spațiile plasate după numele funcției sunt interzise

Dacă există un spațiu după numele funcției, analizorul MySQL poate interpreta numele funcției ca pe un nume de coloană. (Numele funcțiilor nu sunt cuvinte rezervate și le puteți folosi pentru nume de coloană, dacă doriți.) Rezultatul obișnuit este o eroare de sintaxă.

În expresii se pot folosi și valori din coloanele tabelelor. În situația cea mai simplă, când tabelul căruia îi aparține o coloană rezultă evident din context, o referință de coloană se poate rezuma, pur și simplu, la numele coloanei. În fiecare din următoarele instrucțiuni SELECT este specificat un singur tabel; astfel, referințele la coloane sunt lipsite de echivoc:

```
SELECT nume, prenume FROM presedinte
SELECT nume, prenume FROM membru
```

Dacă nu este clară identitatea tabelului care va fi folosit, numele coloanelor pot fi precedate de numele tabelului. Dacă nu este clar nici măcar numele bazei de date care trebuie folosită, numele tabelului poate fi precedat de numele bazei de date. De asemenea, puteți folosi aceste forme mai concrete în contexte lipsite de echivoc, dacă doriți pur și simplu să fiți mai explicit:

```
SELECT
    presedinte.nume, presedinte.prenume,
    membru.nume, membru.prenume
FROM presedinte, membru
WHERE presedinte.nume = membru.nume

SELECT samp_db.elev.nume FROM samp_db.elev
```

În final, puteți combina toate aceste tipuri de valori pentru a forma expresii mai complexe.

Tipuri de operatori

MySQL include numeroase tipuri de operatori care pot fi folosiți pentru a combina termenii expresiilor. Operatorii aritmetici, prezentați în tabelul 2.15, includ operatorii obișnuiți de adunare, scădere, înmulțire și împărțire, precum și operatorul modulo. Operațiile aritmetice se execută folosind valori întregi BIGINT (de 64 biți) pentru opera-

torii +, – și * atunci când ambii operanzi sunt întregi, precum și pentru operatorii / și % atunci când operația este efectuată într-un context unde se așteaptă ca rezultatul să fie un întreg. Rețineți că, dacă o operație implică valori mari, astfel încât rezultatul depășește domeniul pe 64 de biți, veți obține rezultate imprevizibile.

Tabelul 2.15 Operatori aritmetici

Operator	Sintaxă	Semnificație
+	a + b	Adunare; suma operanzilor
-	a - b	Scădere; diferența între operanzi
-	-a	Minus unar; negația operandului
*	a * b	Înmulțire; produsul operanzilor
/	a / b	Împărțire; câtul operanzilor
%	a % b	Modulo; restul rămas după împărțirea operanzilor

Operatorii logici, prezentați în tabelul 2.16, evaluează expresii pentru a determina dacă sunt adevărate (diferite de zero) sau false (zero). MySQL include operatorii „în stil C“ &&, || și ! ca forme alternative pentru AND, OR și NOT. Observați mai ales operatorul ||; ANSI SQL specifică || ca operator de concatenare a șirurilor, dar în MySQL are semnificația unei operații OR (SAU) logice. Dacă executați următoarea interogare și vă așteptați ca aceasta să execute concatenarea șirurilor, veți descoperi cu surprindere că returnează numărul 0:

```
SELECT "abc" || "def"                    ⇒ 0
```

"abc" și "def" sunt convertite în întregi în vederea operației, iar ambele se transformă în 0. În MySQL, trebuie să folosiți funcția CONCAT("abc","def") pentru a efectua concatenarea șirurilor.

Tabelul 2.16 Operatori logici

<table>
<tr><th>Operator</th><th>Sintaxă</th><th>Semnificație</th></tr>
<tr><td>AND, &&</td><td>a AND b, a && b</td><td>Intersecție logică; adevărată dacă ambii operanzi sunt adevărați</td></tr>
<tr><td>OR, ||</td><td>a OR b, a || b</td><td>Reuniune logică; adevărată dacă oricare din operanzi este adevărat</td></tr>
<tr><td>NOT, !</td><td>NOT a, !a</td><td>Negare logică; adevărată dacă operandul este fals</td></tr>
</table>

Operatorii pe bit, prezentați în tabelul 2.17, execută intersecții și reuniuni la nivel de bit, unde fiecare bit al rezultatului este evaluat sub forma intersecției sau reuniunii logice a biților corespunzători ai operanzilor. De asemenea, puteți executa deplasări ale biților la stânga sau la dreapta. Operațiile cu biți sunt efectuate folosindu-se valori întregi BIGINT (pe 64 de biți).

Tabelul 2.17 Operatori pe bit

<table>
<tr><th>Operator</th><th>Sintaxă</th><th>Semnificație</th></tr>
<tr><td>&</td><td>a & b</td><td>Intersecție (AND) la nivel de bit; fiecare bit al rezultatului este setat dacă biții corespunzători ai ambilor operanzi sunt setați</td></tr>
<tr><td>|</td><td>a | b</td><td>Reuniune (OR) la nivel de bit; fiecare bit al rezultatului este setat dacă bitul corespunzător al oricăruia dintre operanzi este setat</td></tr>
<tr><td><<</td><td>a << b</td><td>Deplasare la stânga a lui a cu b poziții de bit</td></tr>
<tr><td>>></td><td>a >> b</td><td>Deplasare la dreapta a lui a cu b poziții de bit</td></tr>
</table>

Operatorii de comparație, prezentați în tabelul 2.18, includ operatori pentru testarea mărimii relative sau a aranjării lexicografice a numerelor și a șirurilor, precum și operatori pentru stabilirea corespondenței cu un model și pentru detectarea valorilor NULL. Operatorul <=> este specific limbajului MySQL și a fost introdus în MySQL 3.23.

Tabelul 2.18 Operatori de comparație

Operator	Sintaxă	Semnificație
=	a = b	Adevărat dacă operanzii sunt egali
!=, <>	a != b, a <> b	Adevărat dacă operanzii sunt diferiți
<	a < b	Adevărat dacă a este mai mic decât b
<=	a <= b	Adevărat dacă a este mai mic sau egal cu b
>=	a >= b	Adevărat dacă a este mai mare sau egal cu b
>	a > b	Adevărat dacă a este mai mare decât b
IN	a IN (b1, b2, ...)	Adevărat dacă a este egal cu oricare din termenii b1, b2, ...
BETWEEN	a BETWEEN b AND c	Adevărat dacă a se află între valorile lui b și c, inclusiv b și c
LIKE	a LIKE b	Corespondență cu modelul în SQL; adevărat dacă a corespunde cu b
NOT LIKE	a NOT LIKE b	Corespondență cu modelul în SQL; adevărat dacă a nu corespunde cu b
REGEXP	a REGEXP b	Corespondență cu o expresie regulată extinsă; adevărată dacă a corespunde cu b
NOT REGEXP	a NOT REGEXP b	Corespondență cu o expresie regulată extinsă; adevărată dacă a nu corespunde cu b
<=>	a <=> b	Adevărat dacă operanzii sunt egali (chiar dacă au valoarea NULL)
IS NULL	a IS NULL	Adevărat dacă operandul este NULL
IS NOT NULL	a IS NOT NULL	Adevărat dacă operandul nu este NULL

Operatorul BINARY este disponibil începând cu MySQL versiunea 3.23 și poate fi folosit pentru conversia unui șir într-un șir binar, astfel încât să fie sensibil la diferența între

majuscule și minuscule în cazul comparațiilor. Prima dintre următoarele comparații nu este sensibilă la diferența între majuscule și minuscule, dar a doua și a treia sunt:

```
"abc" = "Abc"                    1
BINARY "abc" = "Abc"             0
"abc" = BINARY "Abc"             0
```

Nu există nici o conversie corespunzătoare NOT BINARY. Dacă doriți să folosiți o coloană atât într-un context sensibil la diferența între majuscule și minuscule, cât și într-un context fără această caracteristică, folosiți un tip de coloană care nu este sensibil la diferența între majuscule și minuscule și utilizați BINARY pentru acele comparații pe care le doriți sensibile la diferența între majuscule și minuscule.

Comparațiile sunt întotdeauna sensibile la diferența între majuscule și minuscule pentru coloanele pe care le declarați folosind un tip de șir binar (CHAR BINARY, VARCHAR BINARY și tipurile BLOB). Pentru a obține o comparație care nu este sensibilă la diferența între majuscule și minuscule pentru un asemenea tip de coloană, folosiți UPPER() sau LOWER() pentru a converti ambii operanzi la aceeași mărime a literei:

```
UPPER(nume_coloana) < UPPER("Ionescu")
LOWER(nume_coloana) < LOWER("Ionescu")
```

Pentru comparații între șiruri care nu sunt sensibile la diferența între majuscule și minuscule, este posibil ca mai multe caractere să fie considerate echivalente, în funcție de setul dumneavoastră de caractere. De exemplu, E și É s-ar putea să fie tratate identic în cadrul operațiilor de comparație și ordonare. Comparațiile binare (sensibile la diferența între majuscule și minuscule) se execută folosind valorile ASCII ale caracterelor.

Corespondența cu un model vă ajută să căutați valori fără a fi obligat să specificați o valoare literală exactă. MySQL furnizează corespondența cu un model din SQL folosind operatorul LIKE și caracterele de înlocuire % (corespunde oricărei secvențe de caractere) și _ (corespunde oricărui caracter individual). De asemenea, MySQL furnizează corespondența cu un model bazându-se pe operatorul REGEXP și pe expresiile regulate extinse care sunt similare celor folosite în programele UNIX precum grep, sed și vi. Trebuie să folosiți unul dintre acești operatori de stabilire a corespondențelor cu un model pentru a efectua stabilirea corespondenței cu un model; nu puteți folosi operatorul =. Pentru a inversa sensul unei operații de stabilire a corespondenței cu un model, folosiți NOT LIKE sau NOT REGEXP.

Cele două tipuri de stabilire a corespondenței cu un model diferă sub două aspecte importante, dincolo de utilizarea unor operatori diferiți și a unor caractere de model diferite:

- LIKE nu este sensibil la diferența între majuscule și minuscule decât dacă minimum un operand este un șir binar. REGEXP este sensibil la diferența între majuscule și minuscule[10].
- Modelele SQL corespund cu valoarea găsită numai dacă întregul șir corespunde. Expresiile regulate corespund dacă modelul este găsit oriunde în interiorul șirului.

[10] Începând de la versiunea 3.23.4, REGEXP nu mai este sensibil la diferența între majuscule și minuscule decât dacă minimum un operand este un șir binar. – N.A.

Modelele folosite cu operatorul LIKE pot include caracterele de înlocuire % și _. De exemplu, modelul "ele%" corespunde tuturor șirurilor care încep cu "ele":

```
"elevat" LIKE "ele%"             ⇨ 1
"elementar" LIKE "ele%"          ⇨ 1
```

Caracterul de înlocuire % corespunde oricărei secvențe de caractere, inclusiv cu secvența vidă, deci "ele%" corespunde lui "ele":

```
"ele" LIKE "ele%"                ⇨ 1
```

Aceasta mai înseamnă că modelul "%" corespunde oricărui șir, inclusiv șirului vid. Totuși, "%" nu va corespunde valorii NULL. De fapt, orice stabilire de corespondență cu un model care conține un operand NULL va eșua:

```
"ele" LIKE NULL                  ⇨ NULL
NULL LIKE "ele%"                 ⇨ NULL
```

Operatorul LIKE din MySQL nu este sensibil la diferența între majuscule și minuscule, decât dacă unul din operanzii săi este un șir binar. Astfel, "ele%" este echivalent în mod prestabilit atât cu șirul "elevat" cât și cu șirul "Elevat", dar într-o comparație binară va fi echivalent numai cu unul din cele două șiruri:

```
"Elevat" LIKE "ele%"             ⇨ 1
"elevat" LIKE "ele%"             ⇨ 1
BINARY "elevat" LIKE "ele%"      ⇨ 1
BINARY "Elevat" LIKE "ele%"      ⇨ 0
```

Acest lucru nu se întâmplă în cazul operatorului LIKE din ANSI SQL, care este sensibil la diferența între majuscule și minuscule.

Caracterul de înlocuire poate fi specificat oriunde în interiorul modelului. De exemplu, "%at" este echivalent cu "birocrat", "colorat" sau "aristocrat". Pe de altă parte, "%at%" este echivalent cu toate șirurile de mai sus, dar și cu șiruri gen "crater", "atent" sau "straturi".

Celălalt caracter de înlocuire permis pentru LIKE este _, care corespunde oricărui caracter individual. ___ corespunde oricărui șir format din exact trei caractere, "a_r" corespunde cu "aer" și "aur" și chiar cu "a_r", deoarece _ este echivalent cu el însuși.

Pentru a dezactiva semnificația specială a caracterelor de înlocuire % și _, pentru a corespunde instanțelor literale ale acestor caractere, precedați-le cu un backslash (\% sau _):

```
"abc" LIKE "a%c"                 ⇨ 1
"abc" LIKE "a\%c"                ⇨ 0
"a%c" LIKE "a\%c"                ⇨ 1
```

Cealaltă formă folosită în MySQL pentru stabilirea corespondențelor cu un model folosește expresiile regulate. Operatorul este REGEXP, nu LIKE. (RLIKE este un sinonim pentru REGEXP.) Caracterele de model cele mai folosite în cazul expresiilor regulate sunt următoarele:

. corespunde oricărui caracter individual:

```
"abc" REGEXP "a.c"               ⇨ 1
```

`[...]` corespunde oricărui caracter plasat între paranteze drepte. Puteți specifica un domeniu de caractere prin precizarea extremităților domeniului, separate printr-o liniuță (-). Pentru a nega sensul clasei (astfel încât să corespundă tuturor caracterelor care nu se află pe listă), specificați `^` ca prim caracter al clasei:

```
"abc" REGEXP "[a-z]"          ⇨ 1
"abc" REGEXP "[^a-z]"         ⇨ 0
```

`*` corespunde oricărui număr de elemente la fel cu cel anterior semnului. De exemplu, `x*` corespunde oricărui număr de caractere `x`:

```
"abcdef" REGEXP "a.*f"        ⇨ 1
"abc" REGEXP "[0-9]*abc"      ⇨ 1
"abc" REGEXP "[0-9][0-9]*"    ⇨ 0
```

„Orice număr" înseamnă și zero apariții, motiv pentru care a doua expresie este adevărată.

`^mod` și `mod$` ancorează o corespondență cu un model astfel încât modelul `mod` corespunde numai atunci când survine la începutul sau la sfârșitul unui șir, iar `^mod$` corespunde numai dacă `mod` corespunde întregului șir:

```
"abc" REGEXP "b"              ⇨ 1
"abc" REGEXP "^b"             ⇨ 0
"abc" REGEXP "b$"             ⇨ 0
"abc" REGEXP "^abc$"          ⇨ 1
"abcd" REGEXP "^abc$"         ⇨ 0
```

Modelul `REGEXP` poate fi luat din coloana unui tabel, deși această operație va fi mai lentă decât un model de tip constantă, în cazul în care coloana conține numeroase valori diferite. Modelul trebuie examinat și convertit în formă internă de fiecare dată când valoarea din coloană se modifică.

Stabilirea corespondențelor cu expresiile regulate din MySQL dispune și de alte caractere model speciale. Vezi Anexa C pentru mai multe informații.

Precedența operatorilor

Când MySQL evaluează o expresie, examinează operatorii pentru a determina ordinea în care trebuie să grupeze termenii expresiei. Unii operatori au o precedență mai ridicată, adică sunt mai „puternici" decât alții, în sensul că sunt evaluați înaintea altora. De exemplu, înmulțirea și împărțirea au o precedență mai ridicată decât adunarea și scăderea. Următoarele două expresii sunt echivalente, deoarece `*` și `/` sunt evaluate înainte de `+` și `-`:

```
1 + 2 * 3 - 4 / 5             ⇨ 6.2
1 + 6 - .8                    ⇨ 6.2
```

Precedența operatorilor este prezentată în lista următoare, în ordine descrescătoare.

Operatorii menționați pe aceeași linie au aceeași precedență. Operatorii cu un nivel de precedență mai ridicat sunt evaluați înaintea operatorilor cu un nivel de precedență mai redus.

```
BINARY
NOT !
- (minus unar)
```



```
* / %
+ -
<< >>
&
|
< <= = <=> != <> >= > IN IS LIKE REGEXP RLIKE
BETWEEN
AND &&
OR ||
```

Puteți folosi paranteze pentru a ignora precedența operatorilor și pentru a modifica ordinea în care sunt evaluați termenii unei expresii:

```
1 + 2 * 3 - 4 / 5             ⇨ 6.2
(1 + 2) * (3 - 4) / 5         ⇨ -0.6
```

Valorile NULL în expresii

Fiți atent când folosiți valoarea `NULL` în expresii, deoarece rezultatul s-ar putea să nu fie întotdeauna cel scontat. Îndrumările următoare vă vor ajuta să evitați surprizele.

Dacă specificați `NULL` ca operand pentru orice operator aritmetic sau la nivel de bit, rezultatul este `NULL`:

```
1 + NULL                      ⇨ NULL
1 | NULL                      ⇨ NULL
```

Dacă folosiți `NULL` cu un operator logic, valoarea `NULL` este considerată falsă:

```
1 AND NULL                    ⇨ 0
1 OR NULL                     ⇨ 1
0 AND NULL                    ⇨ 0
0 OR NULL                     ⇨ 0
```

`NULL` ca operand pentru orice operator de comparație produce un rezultat `NULL`, cu excepția operatorilor `<=>`, `IS NULL` și `IS NOT NULL`, care sunt special concepuți pentru manipularea valorilor `NULL`:

```
1 = NULL                      ⇨ NULL
NULL = NULL                   ⇨ NULL
1 <=> NULL                    ⇨ 0
NULL <=> NULL                 ⇨ 1
1 IS NULL                     ⇨ 0
NULL IS NULL                  ⇨ 1
```

În general, funcțiile returnează `NULL` dacă primesc argumente `NULL`, cu excepția acelor funcții concepute pentru a manipula argumente `NULL`. De exemplu, `IFNULL()` poate manipula argumente `NULL` și returnează „adevărat" sau „fals" în consecință. `STRCMP()` presupune argumente non-`NULL`; dacă descoperă că i-ați transferat un argument `NULL`, returnează `NULL`, nu „adevărat" sau „fals".

În cadrul operațiilor de sortare, valorile `NULL` se grupează la un loc. `NULL` este plasat, după sortare, înaintea tuturor valorilor non-`NULL` (inclusiv șirul vid) în cazul unei sortări ascendente, respectiv după toate aceste valori la o sortare descendentă.

Conversia de tip

MySQL execută automat conversii masive de tip, în conformitate cu genul de operație pe care o efectuați, ori de câte ori o valoare de un tip este folosită într-un context care necesită o valoare de un alt tip. Conversia de tip se poate produce din oricare din următoarele motive:

- Conversia operanzilor la un tip adecvat pentru evaluarea unui operator
- Conversia argumentului unei funcții la un tip scontat de funcție
- Conversia unei valori pentru repartizarea într-o coloană de tabel care este de un alt tip

Expresia următoare necesită conversia de tip. Expresia constă din operatorul de adunare și doi operanzi, `1` și `"2"`:

```
1 + "2"
```

Operanzii sunt de tipuri diferite (număr și șir), deci MySQL îl convertește pe unul pentru a-i aduce la un numitor comun. Dar care este operandul care trebuie modificat? În acest caz, + este un operator numeric, deci MySQL dorește ca operanzii să fie numere și convertește șirul `"2"` în numărul `2`. Apoi, evaluează expresia pentru a produce rezultatul 3.

Iată un alt exemplu. Funcția `CONCAT()` concatenează șirurile pentru a produce drept rezultat un șir mai lung. Pentru aceasta, funcția își interpretează argumentele sub formă de șiruri, indiferent de tipul acestora. Dacă îi transferați o mulțime de numere, `CONCAT()` le va converti în șiruri, după care va returna rezultatul concatenării acestora:

```
CONCAT(1,2,3)                    ⇨ "123"
```

Dacă apelul la funcția `CONCAT` face parte dintr-o expresie mai mare, vor mai avea loc și alte conversii de tip. Să luăm următoarea expresie și rezultatul ei:

```
REPEAT('X',CONCAT(1,2,3)/10)     ⇨ "XXXXXXXXXXXX"
```

`CONCAT(1,2,3)` produce șirul `"123"`. Expresia `"123"/10` este convertită în `123/10`, deoarece împărțirea este un operator aritmetic. Rezultatul aceste expresii va fi `12.3` într-un context cu virgulă mobilă, dar funcția `REPEAT()` presupune un număr întreg de repetări, deci se execută o împărțire întreagă pentru a se obține rezultatul 12. Apoi, funcția `REPEAT('X',12)` are ca rezultat un șir format din 12 caractere `'X'`.

Un principiu general care trebuie reținut este acela că MySQL încearcă să convertească valorile în tipul cerut de o expresie, în loc de a genera o eroare. În funcție de context, MySQL va converti valorile fiecăreia din cele trei categorii generale (numere, șiruri sau date și ore) în valori ale oricăreia din celelalte categorii. Totuși, valorile nu pot fi întotdeauna convertite dintr-un tip în altul. Dacă o valoare care trebuie convertită într-un tip dat nu este o valoare admisă pentru tipul respectiv, conversia eșuează. Conversia în numere a unor obiecte precum `"abc"` care nu prezintă un aspect numeric are ca rezultat valoarea `0`.

Conversia în tipuri dată și oră a unor obiecte care au aspectul unor date sau ore are ca rezultat valoarea „zero" pentru tipul respectiv. De exemplu, conversia șirului `"abc"` într-o dată are ca rezultat data de tip zero `"0000-00-00"`. Pe de altă parte, orice valoare poate fi tratată sub forma unui șir, deci în general conversia unei valori într-un șir nu constituie o problemă.

De asemenea, MySQL execută și conversii minore de tip. Dacă folosiți o valoare cu virgulă mobilă într-un context întreg, valoarea este convertită (cu rotunjire). Conversia în cealaltă direcție este de asemenea posibilă; un întreg poate fi folosit fără probleme ca număr în virgulă mobilă.

Constantele hexazecimale sunt tratate ca șiruri, cu excepția situațiilor când contextul indică cu claritate un număr. În contextele șir, fiecare pereche de cifre hexazecimale este convertită într-un caracter și rezultatul este folosit ca șir. Următoarele exemple ilustrează acest principiu:

```
0x61                    ⇨ "a"
0x61 + 0                ⇨ 97
CONCAT(0X61)            ⇨ "a"
CONCAT(0X61 + 0)        ⇨ "97"
```

Același principiu de interpretare se aplică și în cazul comparațiilor: o constantă hexazecimală este tratată ca șir, cu excepția cazurilor când este comparată cu un număr:

```
10 = 0x0a               ⇨ 1
10 = 0x09               ⇨ 0
"\n" = 0x0a             ⇨ 1
"\n" = 0x0a + 0         ⇨ 0
("\n" = 0x0a) + 0       ⇨ 1
```

Unii operatori forțează conversia operanzilor în tipul așteptat de operator, indiferent de tipul operanzilor. Operatorii aritmetici sunt un exemplu în acest sens; ei așteaptă numere, iar operanzii sunt convertiți în consecință:

```
3 + 4                   ⇨ 7
"3" + 4                 ⇨ 7
"3" + "4"               ⇨ 7
```

Nu este suficient ca un șir să conțină pur și simplu un număr pe undeva. MySQL nu caută în tot șirul, în speranța că va găsi un număr; caută numai la început. Dacă un șir nu are nici o parte numerică la început, rezultatul conversiei este `0`.

```
"1973-2-4" + 0          ⇨ 1973
"12:14:01" + 0          ⇨ 12
"23-valea" + 0          ⇨ 23
"-23-valea" + 0         ⇨ -23
"carbon-14" + 0         ⇨ 0
```

Rețineți că regula de conversie șir-număr din MySQL s-a modificat începând de la versiunea 3.23. Anterior acelei versiuni, șirurile cu aspect numeric au fost convertite în valori întregi, cu rotunjire. De la versiunea 3.23, acestea vor fi convertite în valori cu virgulă mobilă:

```
"-428.9" + 0            ⇨ -429 (MySQL < 3.23)
"-428.9" + 0            ⇨ -428.9 (MySQL  3.23)
```

Operatorii logici și cu biți sunt chiar mai stricți decât operatorii aritmetici. Aceștia impun ca operanzii să nu fie numai numerici, ci întregi, iar conversia de tip se execută în consecință. Aceasta înseamnă că un număr cu virgulă mobilă cum este 0,3 nu este considerat adevărat, chiar dacă este diferit de zero; aceasta deoarece, atunci când este

convertit la un întreg, rezultatul este 0. În expresiile următoare, operanzii nu sunt considerați adevărați decât atunci când au o valoare cel puțin egală cu 1.

```
0.3 OR .04        ⇨ 0
1.3 OR .04        ⇨ 1
0.3 AND .04       ⇨ 0
1.3 AND .04       ⇨ 0
1.3 AND 1.04      ⇨ 1
```

Acest tip de conversie se produce și în cazul funcției IF(), care presupune ca primul argument să fie un întreg. Pentru a testa în mod adecvat valorile cu virgulă mobilă, cel mai bine este să folosiți o comparație explicită. În caz contrar, valorile mai mici decât 1 vor fi considerate false:

```
IF(1.3,"non-zero","zero")     ⇨ "non-zero"
IF(0.3,"non-zero","zero")     ⇨ "zero"
IF(0.3>0,"non-zero","zero")   ⇨ "non-zero"
```

Operatorii de stabilire a corespondențelor cu un model se așteaptă să opereze cu șiruri. Aceasta înseamnă că puteți folosi operatorii MySQL de stabilire a corespondențelor cu un model folosind numere ca operanzi, deoarece aceștia vor converti numerele în șiruri, în încercarea de a găsi o corespondență!

```
12345 LIKE "1%"        ⇨ 1
12345 REGEXP "1.*5"    ⇨ 1
```

Operatorii de comparație a mărimii (<, <=, = și așa mai departe) sunt sensibili la context; cu alte cuvinte, sunt evaluați în conformitate cu tipul operanzilor lor. Expresia următoare compară operanzii prin mijloace numerice, deoarece ambii operanzi sunt numere:

```
2 < 11      ⇨ 1
```

Această expresie implică o comparație între șiruri (lexicografică), deoarece ambii operanzi sunt șiruri:

```
"2" < "11"      ⇨ 0
```

În următoarele comparații, tipurile sunt combinate, deci MySQL le compară sub formă de numere. Ca rezultat, ambele expresii sunt adevărate:

```
"2" < 11      ⇨ 1
2 < "11"      ⇨ 1
```

În comparații, MySQL convertește operanzii conform necesităților, în conformitate cu următoarele reguli:

- În afara celor care implică operatorul <=>, comparațiile care implică NULL au ca rezultat NULL. (Operatorul <=> este similar cu =, cu excepția faptului că afirmația NULL <=> NULL este adevărată.)
- Dacă ambii operanzi sunt șiruri, atunci vor fi comparați lexicografic ca șiruri. Comparațiile între șiruri sunt efectuate folosind setul de caractere aflat în vigoare pe server.
- Dacă ambii operanzi sunt întregi, vor fi comparați numeric ca întregi.
- Constantele hexazecimale care nu sunt comparate cu un număr sunt comparate ca șiruri binare.

- Dacă oricare dintre operanzi este o valoare TIMESTAMP sau DATETIME și celălalt este o constantă, operanzii sunt comparați ca valori TIMESTAMP. Acest lucru se face cu scopul de a determina comparațiile să funcționeze mai bine pentru aplicațiile ODBC.
- În caz contrar, operanzii sunt comparați numeric ca valori cu virgulă mobilă. Rețineți că aici este inclus și cazul comparației unui șir cu un număr. Șirul este convertit la un număr, ceea ce are ca rezultat valoarea 0 dacă șirul nu arată ca un număr. De exemplu, "14.3" este convertit la 14.3, dar "L4.3" este convertit la zero.

Reguli de interpretare a datei și a orei

MySQL convertește liber șirurile și numerele în valori de tip dată și oră, conform cerințelor contextului unei expresii, și viceversa. Valorile de tip dată și oră sunt convertite în numere într-un context numeric; numerele sunt convertite în date sau ore într-un context de tip dată sau oră. Această conversie la o valoare de tip dată sau oră se produce atunci când repartizați o valoare într-o coloană de tip dată sau oră, respectiv când o funcție necesită o valoare de tip dată sau oră. În comparații, regula generală este ca valorile de tip dată și oră să fie comparate ca șiruri.

Dacă tabelul tabelul_meu conține o coloană DATE denumită data_col, următoarele instrucțiuni sunt echivalente:

```
INSERT INTO tabelul_meu SET data_col = "1997-04-13"
INSERT INTO tabelul_meu SET data_col = "19970413"
INSERT INTO tabelul_meu SET data_col = 19970413
```

În exemplele următoare, argumentul funcției TO_DAYS() este interpretat ca fiind una și aceeași valoare pentru toate cele trei expresii:

```
TO_DAYS("1997-04-10")     ⇨ 729489
TO_DAYS("19970410")       ⇨ 729489
TO_DAYS(19970410)         ⇨ 729489
```

Testarea și forțarea conversiei de tip

Pentru a examina modul de efectuare a conversiei de tip într-o expresie, folosiți programul mysql pentru a emite o interogare SELECT care evaluează expresia:

```
mysql> SELECT 0x41, 0x41 + 0;
```

0x41	0x41 + 0
A	65

Așa cum vă puteți imagina, am făcut o mulțime de lucruri ca acesta în timpul scrierii capitolului de față!

Testarea evaluării expresiilor este importantă mai ales pentru instrucțiunile precum DELETE sau UPDATE care modifică înregistrări, deoarece doriți să fiți sigur că modificați numai rândurile scontate. O modalitate de a verifica o expresie este de a rula o instrucțiune SELECT preliminară cu aceeași clauză WHERE pe care urmează să o folosiți pentru instrucțiunea DELETE sau UPDATE, pentru a verifica în ce măsură clauza selectează rândurile adecvate. Să presupunem că tabelul tabelul_meu are o coloană CHAR denumită char_col, care conține următoarele valori:

```
"abc"
"def"
"00"
"ghi"
"jkl"
"00"
"mno"
```

Dacă se dau aceste valori, care este efectul următoarei interogări?

```
DELETE FROM tabelul_meu WHERE char_col = 00
```

Efectul *scontat* este probabil ștergerea celor două rânduri care conțin valoarea `"00"`. Efectul real este ștergerea tuturor rândurilor; o surpriză neplăcută. Aceasta se întâmplă ca o consecință a regulilor de comparație folosite în MySQL. `char_col` este o coloană de tip șir, dar `00` nu este inclus între ghilimele, deci este tratat ca număr. Conform regulilor de comparație din MySQL, o comparație între un șir și un număr este considerată ca o comparație între două numere. La executarea interogării `DELETE`, fiecare valoare din coloana `char_col` este convertită într-un număr și comparată cu 0: `"00"` se convertește la 0, dar tot la 0 se vor converti și celelalte șiruri care nu au aspect numeric. Ca atare, clauza `WHERE` este adevărată pentru fiecare rând, iar instrucțiunea `DELETE` șterge tabelul. Evident, aceasta este o situație unde ar fi fost prudent să se testeze clauza `WHERE` cu o instrucțiune `SELECT` înaintea execuției instrucțiunii `DELETE`, deoarece aceasta v-ar fi arătat că expresia selectează prea multe rânduri:

```
mysql> SELECT char_col FROM tabelul_meu WHERE char_col = 00;
+----------+
| char_col |
+----------+
| "abc"    |
| "def"    |
| "00"     |
| "ghi"    |
| "jkl"    |
| "00"     |
| "mno"    |
+----------+
```

Când nu sunteți sigur cu privire la modul în care va fi folosită o valoare, puteți dori să exploatați mecanismul MySQL de evaluare a expresiilor, pentru a forța conversia la o valoare de un anumit tip:

- Adăugați `+0` sau `+0.0` la un termen pentru a forța conversia la o valoare numerică:

```
0x65          ⇨ "e"
0x65 + 0      ⇨ 101
0x65 + 0.0    ⇨ 101.0
```

- Folosiți `CONCAT()` pentru a transforma o valoare într-un șir:

```
14            ⇨ 14
CONCAT(14)    ⇨ "14"
```

- Utilizați `ASCII()` pentru a obține valoarea ASCII a unui caracter:

```
"A"           ⇨ "A"
ASCII("A")    ⇨ 65
```


- Folosiți `DATE_ADD()` pentru a forța tratarea unui șir sau a unui număr ca dată:

```
19990101                               ⇨ 19990101
DATE_ADD(19990101, INTERVAL 0 DAY)     ⇨ "1999-01-01"
"19990101"                             ⇨ 19990101
DATE_ADD("19990101", INTERVAL 0 DAY)   ⇨ "1999-01-01"
```

Conversia valorilor din afara domeniului sau a valorilor nepermise

Principiul de bază este: semeni vânt, culegi furtună. Dacă nu vă verificați datele în prealabil înainte de a le stoca, s-ar putea să nu vă placă rezultatele obținute. Acestea fiind zise, iată câteva principii generale care descriu modalitățile folosite de MySQL pentru manipularea valorilor situate în afara domeniului sau necorespunzătoare din alte motive:

- Pentru coloane numerice sau de tip `TIME`, valorile care se află în afara domeniului admis sunt „tăiate" până la valoarea celei mai apropiate extremități a domeniului, iar valoarea rezultantă este stocată.
- Pentru coloane de tip șir altele decât `ENUM` sau `SET`, șirurile prea lungi sunt trunchiate, pentru a se încadra în lungimea maximă a coloanei. Repartizările într-o coloană `ENUM` sau `SET` depind de valorile care sunt menționate ca permise la declararea coloanei. Dacă repartizați într-o coloană `ENUM` o valoare care nu este menționată ca membru al enumerării, în locul acesteia este repartizat membrul „eroare" (adică șirul vid care corespunde membrului cu valoare zero). Dacă repartizați coloanei `SET` o valoare care conține sub-șiruri care nu sunt menționate ca membri ai setului, șirurile respective sunt eliminate și coloana primește o valoare care constă din membrii rămași.
- Pentru coloanele de tip dată sau oră, valorile incorecte sunt convertite în valorile „zero" adecvate pentru tipul respectiv (vezi tabelul 2.11). Pentru coloanele de tip dată și oră altele decât `TIME`, valorile care se găsesc în afara domeniului aferent tipului de coloană pot fi convertite la valoarea „zero", la `NULL` sau la o altă valoare. (Cu alte cuvinte, rezultatele sunt imposibil de anticipat.)

Aceste conversii sunt raportate ca avertismente pentru instrucțiunile `ALTER TABLE`, `LOAD DATA`, `UPDATE`, precum și pentru instrucțiunile `INSERT` cu mai multe rânduri. În clientul `mysql`, aceste informații sunt afișate în linia de stare care este raportată pentru o interogare. Într-un limbaj de programare, vă puteți procura aceste informații prin alte mijloace. Dacă folosiți interfața API din MySQL pentru C, puteți apela funcția `mysql_info()`. Cu ajutorul interfeței API pentru modulul DBI din Perl, puteți folosi atributul `mysql_info()` al conexiunii dumneavoastră cu baza de date. Informațiile furnizate reprezintă numărul de avertismente. Pentru a vedea care au fost rândurile modificate, puteți emite o interogare `SELECT ... INTO OUTFILE` și comparați rezultatul cu rândurile originale.

CAPITOLUL 3

Sintaxa și utilizarea SQL în MySQL

Pentru a putea întreține comunicații cu serverul MySQL este necesar să „vorbiți" fluent limbajul MySQL. De exemplu, când folosiți un program cum este clientul `mysql`, acesta funcționează cu precădere ca un mijloc care vă permite să trimiteți instrucțiuni SQL către server în vederea executării acestora. De asemenea, trebuie să cunoașteți limbajul SQL dacă doriți să scrieți programe care folosesc interfața MySQL furnizată de limbajul dumneavoastră de programare, deoarece veți continua să comunicați cu serverul transmițându-i instrucțiuni SQL.

Capitolul 1, „Introducere în MySQL și SQL", a prezentat o introducere în stil „manual" pentru numeroase dintre caracteristicile sistemului MySQL. Acest capitol se bazează pe materialul respectiv pentru a intra în detalii privind numeroase domenii ale sistemului SQL implementate de către MySQL. Se discută despre modul de referire la elementele bazelor de date, inclusiv regulile de denumire și restricțiile care se aplică privind sensibilitatea la diferența între majuscule și minuscule. De asemenea, sunt descrise multe dintre instrucțiunile SQL mai importante, precum cele pentru crearea și distrugerea bazelor de date, a tabelelor și a indexurilor, instrucțiunile pentru regăsirea datelor folosind unirile și instrucțiunile care furnizează informații despre bazele de date și tabelele dumneavoastră. Expunerea scoate în evidență unele dintre extensiile aduse de MySQL limbajului SQL standard.

Ce există și ce lipsește în MySQL?

Instrucțiunile SQL din MySQL pot fi grupate în mai multe categorii largi, așa cum se poate vedea în figura 3.1. În acest capitol, vom discuta despre instrucțiunile din primele patru categorii prezentate în această figură. Unele dintre utilitarele MySQL furnizează ceea ce este, în esență, o interfață de tip linie de comandă cu anumite instrucțiuni SQL. De exemplu, `mysqlshow` este o interfață cu instrucțiunea `SHOW COLUMNS`. Acolo unde este posibil, aceste echivalențe sunt descrise și în capitolul de față.

Dintre instrucțiunile care nu au fost prezentate aici, numeroase vor fi discutate în alte capitole. De exemplu, instrucțiunile `GRANT` și `REVOKE` pentru configurarea privilegiilor de utilizator sunt prezentate în capitolul 11, „Administrarea generală a sistemului MySQL". Sintaxa de invocare a tuturor instrucțiunilor este prezentată în Anexa D, „Referință de sintaxă SQL". De asemenea, trebuie să consultați manualul de referință MySQL pentru informații suplimentare, mai ales pentru modificările efectuate în versiunile mai recente ale limbajului MySQL.

Crearea, ștergerea și selectarea bazelor de date
CREATE DATABASE DROP DATABASE USE
Crearea, modificarea și ștergerea tabelelor și a indexurilor
ALTER TABLE CREATE INDEX CREATE TABLE DROP INDEX DROP TABLE
Obținerea de informații despre baze de date, tabele și interogări
DESCRIBE EXPLAIN SHOW
Selectarea informațiilor din tabele
SELECT
Modificarea informațiilor din tabele
DELETE INSERT LOAD DATA OPTIMIZE TABLE REPLACE UPDATE
Instrucțiuni administrative
FLUSH GRANT KILL REVOKE
Alte instrucțiuni
CREATE FUNCTION DROP FUNCTION LOCK TABLES SET UNLOCK TABLES

Figura 3.1 Instrucțiuni SQL acceptate de MySQL.

Secțiunea finală a capitolului descrie ceea ce *nu* conține MySQL, în speță caracteristicile care îi lipsesc. Acestea sunt funcționalități care se găsesc în alte sisteme de baze de date, dar nu în MySQL. Printre aceste caracteristici se numără subselectări, tranzacții, integritate referențială, declanșatoare, proceduri stocate și vederi. Aceste omisiuni înseamnă cumva că MySQL nu este un sistem „autentic" de baze de date? Unii cred asta, dar ca

răspuns m-aş limita să observ că lipsa acestor caracteristici în MySQL n-a împiedicat folosirea sistemului de către numeroşi utilizatori. Aceasta probabil deoarece, pentru multe aplicaţii, lipsa acestor caracteristici nu contează. În alte situaţii, există metode pentru compensarea unei facilităţi care lipseşte. Lipsa ştergerilor în cascadă, de exemplu, înseamnă că trebuie să emiteţi o interogare suplimentară în aplicaţiile dumneavoastră atunci când ştergeţi înregistrări dintr-un tabel. Lipsa suportului pentru tranzacţii poate să nu conteze prea mult pentru dumneavoastră, dacă vi se pare suficientă capacitatea sistemului MySQL de a grupa instrucţiuni pentru o execuţie neîntreruptă, delimitându-le prin instrucţiuni `LOCK TABLES` şi `UNLOCK TABLES`.

(Adevărata problemă în acest caz nu este, de fapt, lipsa tranzacţiilor; este lipsa unei reveniri automate pentru anularea tuturor instrucţiunilor, dacă vreuna din ele eşuează. Dacă aveţi aplicaţii cu necesităţi precum aceea de efectuare a unor tranzacţii financiare complexe, care implică numeroase instrucţiuni cu interblocare care trebuie executate în grup sau deloc, atunci vă puteţi orienta spre un sistem de baze de date cu facilităţi de înregistrare, respectiv anulare a înregistrării, precum Postgres.) Unele caracteristici lipsesc pur şi simplu deoarece nu au fost încă implementate. De exemplu, subselecţiile nu sunt prezente în MySQL în timpul scrierii acestei cărţi, dar sunt planificate a fi incluse în versiunea 3.24, care este posibil să fi apărut deja în momentul în care dumneavoastră citiţi rândurile de faţă.

Reguli de denumire în MySQL

Aproape fiecare instrucţiune SQL se referă, într-un fel sau altul, la o bază de date sau la elementele sale constitutive. Această secţiune descrie regulile sintactice pentru referirea la bazele de date, la tabele, la coloane, indexuri şi aliasuri. Numele ţin cont de considerente privind sensibilitatea la diferenţa între majuscule şi minuscule, care sunt de asemenea descrise.

Referirea la elementele din bazele de date

Când folosiţi nume pentru a vă referi la elemente ale bazelor de date, sunteţi limitat de numărul de caractere pe care le puteţi folosi şi la lungimea pe care o pot avea numele. Forma numelor depinde şi de contextul în care le folosiţi:

- **Caractere permise în nume.** Numele pot fi constituite din orice caractere alfanumerice din setul de caractere folosit de server, plus caracterele _ şi $. Numele pot începe cu orice caracter care este permis într-un nume, inclusiv o cifră. Totuşi, un nume nu poate fi compus numai din cifre, deoarece astfel s-ar confunda, practic, cu un număr. Posibilitatea furnizată de MySQL de a începe un nume cu un număr este neobişnuită. Dacă folosiţi un asemenea nume, fiţi foarte atent la numele care conţin un `e` sau un `E`, deoarece aceste caractere pot duce la expresii ambigui. `23e + 14` înseamnă coloana `23e` plus `14`, dar ce ne facem cu `23e+14`? Înseamnă acelaşi lucru sau este un număr scris folosind notaţia ştiinţifică?
- **Lungimea numelui.** Numele bazelor de date, ale tabelelor, coloanelor şi indexurilor pot avea maximum 64 de caractere lungime. Numele aliasurilor pot avea maximum 256 de caractere lungime.

- **Calificatori de nume.** Pentru a vă referi la o bază de date, pur şi simplu specificaţi numele acesteia:

```
USE nume_baza_de_date
SHOW TABLES FROM nume_baza_de_date
```

Pentru a face referire la un tabel, aveţi două opţiuni. Un nume de tabel complet determinat constă din numele bazei de date şi numele tabelului:

```
SHOW TABLES FROM nume_baza_de_date.nume_tabel
SELECT * FROM nume_baza_de_date.nume_tabel
```

Un nume de tabel singur face referire la un tabel din baza de date curentă (prestabilită). Dacă `samp_db` este baza de date prestabilită, următoarele instrucţiuni sunt echivalente:

```
SELECT * FROM membru
SELECT * FROM samp_db.membru
```

Pentru a face referire la o coloană, există trei opţiuni: nume complet determinat, nume parţial determinat şi nume nedeterminat. Un nume complet determinat (scris sub forma *`nume_baza_de_date.nume_tabel.nume_coloana`*) este complet precizat. Un nume parţial determinat (scris sub forma *`nume_tabel.nume_coloana`*) se referă la o coloană din tabelul desemnat. Un nume nedeterminat (scris sub forma simplă *`nume_coloana`*) se referă la tabelul indicat de contextul înconjurător. Următoarele două interogări fac referire la aceleaşi nume de coloană, dar contextul furnizat de clauzele `FROM` indică tabelele din care se vor selecta coloanele:

```
SELECT nume, prenume FROM presedinte
SELECT nume, prenume FROM membri
```

De regulă, precizarea unor nume complet determinate nu este necesară, deşi este întotdeauna permisă dacă doriţi acest lucru. Dacă selectaţi o bază de date cu o instrucţiune `USE`, baza de date respectivă devine bază de date prestabilită şi este implicită în fiecare referinţă nedeterminată la un tabel. Dacă folosiţi o instrucţiune `SELECT` care face referire la un singur tabel, tabelul respectiv este implicit pentru fiecare referire la coloană din instrucţiune. Este necesar să precizaţi numele numai atunci când un tabel sau o bază de date nu pot fi determinate din context. Iată câteva situaţii în care poate apărea echivocul:

- *Interogări care fac referire la tabele din mai multe baze de date.* Orice tabel care nu se găseşte în baza de date prestabilită trebuie desemnat folosind forma *`nume_baza_de_date.nume_tabel`*, pentru a permite sistemului MySQL să identifice baza de date în care trebuie să caute pentru a găsi tabelul.
- *Interogări care selectează o coloană din mai multe tabele*, unde mai multe tabele conţin o coloană cu numele respectiv.

Sensibilitatea instrucţiunilor SQL la diferenţa între majuscule şi minuscule

Sensibilitatea instrucţiunilor SQL la diferenţa între majuscule şi minuscule variază pentru diferitele componente ale unei instrucţiuni; de asemenea, mai poate depinde de entitatea la care faceţi referire şi la sistemul de operare pe care rulează serverul:

- **Cuvinte cheie și nume de funcții SQL.** Cuvintele cheie și numele de funcții nu sunt sensibile la diferența între majuscule și minuscule. Ele pot fi scrise folosindu-se orice mărime de literă. Următoarele instrucțiuni sunt echivalente:

```
SELECT NOW()
select now()
sElEcT nOw()
```

- **Numele bazelor de date și ale tabelelor.** Bazele de date și tabelele din MySQL corespund cataloagelor și fișierelor din sistemul de fișiere de bază al gazdei serverului. În consecință, sensibilitatea numelor bazelor de date și a numelor tabelelor la diferența între majuscule și minuscule depinde de modul de tratare a numelor fișierelor de către sistemul de operare al gazdei respective. Un server care rulează sub UNIX tratează numele bazelor de date și ale tabelelor ca sensibile la diferența între majuscule și minuscule, deoarece numele de fișiere UNIX prezintă această sensibilitate. Numele de fișiere Windows nu sunt sensibile la diferența între majuscule și minuscule, deci un server care rulează sub Windows nu tratează numele bazelor de date și ale tabelelor ca fiind sensibile la diferența între majuscule și minuscule.

  Trebuie să fiți la curent cu această caracteristică dacă creați o bază de date pe un server UNIX și pe care într-o zi s-ar putea să o mutați pe un server Windows: dacă creați două tabele denumite `abc` și `ABC`, acestea vor fi imposibil de diferențiat pe un calculator Windows. O modalitate de a evita transformarea acestui aspect într-o problemă este de a selecta o anumită dimensiune a literelor (minuscule, de exemplu) și să creați întotdeauna nume de baze de date și de tabele folosind acea dimensiune de literă. Apoi, mărimea literei numelor nu va mai fi o problemă dacă mutați baza de date pe un alt server.

- **Nume de coloane și de indexuri.** Numele coloanelor și ale indexurilor în MySQL nu sunt sensibile la diferența între majuscule și minuscule. Următoarele interogări sunt echivalente:

```
SELECT nume FROM elev
SELECT NUME FROM elev
SELECT NuMe FROM elev
```

- **Numele aliasurilor.** Aliasurile sunt sensibile la diferența între majuscule și minuscule. Puteți specifica un alias folosind orice dimensiune de literă (majuscule, minuscule sau combinate), dar trebuie ca toate referirile la alias din interogare să folosească aceeași mărime de literă.

Indiferent dacă un nume de bază de date, tabel sau alias este sau nu sensibil la diferența între majuscule și minuscule, trebuie să faceți referire la un nume dat al oricăreia dintre aceste categorii folosind aceeași dimensiune a literei pe parcursul întregii interogări. Acest lucru nu este valabil pentru cuvintele cheie SQL, pentru numele de funcții sau numele coloanelor și al indexurilor, la care se poate face referire cu dimensiuni de literă variabile pe parcursul unei interogări. Natural, interogarea va fi mai lizibilă dacă folosiți o aceeași mărime de literă în loc de stilul „scrisoare anonimă" (`SeLeCt NuMe FrOm ...`).

Crearea, ștergerea și selectarea bazelor de date

MySQL furnizează trei instrucțiuni la nivelul bazelor de date: `CREATE DATABASE` pentru crearea bazelor de date, `DROP DATABASE` pentru eliminarea lor, respectiv `USE` pentru selectarea unei baze de date prestabilite.

Instrucțiunea CREATE DATABASE

Crearea unei baze de date este o operație simplă; pur și simplu denumiți baza de date într-o instrucțiune `CREATE DATABASE`:

```
CREATE DATABASE nume_baza_de_date
```

Restricțiile sunt acelea că numele trebuie să fie permis, baza de date trebuie să nu existe deja, iar dumneavoastră trebuie să aveți suficiente privilegii pentru a o crea.

Instrucțiunea DROP DATABASE

Ștergerea unei baze de date este la fel de simplă ca și crearea acesteia, presupunând că aveți suficiente privilegii:

```
DROP DATABASE nume_baza_de_date
```

Totuși, instrucțiunea `DROP DATABASE` nu este ceva ce trebuie folosit cu un sentiment de genul „După mine, potopul!". Instrucțiunea elimină baza de date și toate tabelele din interiorul acesteia. După ce ați șters o bază de date, s-a terminat cu ea. Cu alte cuvinte, nu încercați această instrucțiune numai pentru a vedea cum funcționează. Dacă administratorul dumneavoastră a efectuat sistematic copii de siguranță ale bazei de date, vă veți putea recupera baza de date. Dar vă pot garanta că nici un administrator nu va manifesta înțelegere când îi veți spune: „A, păi mă jucam cu `DROP DATABASE` ca să văd ce se-ntâmplă, și, ăăă... îmi puteți restaura și mie baza de date?"

Rețineți că o bază de date este reprezentată printr-un catalog aflat sub catalogul de date. Dacă ați inserat în acel catalog fișiere care nu reprezintă tabele, acele fișiere nu vor fi șterse de instrucțiunea `DROP DATABASE`. În acel caz, nici catalogul bazei de date nu este eliminat.

Instrucțiunea USE

Instrucțiunea `USE` selectează o bază de date pentru a o transforma în bază de date curentă (prestabilită) pentru o conexiune dată cu serverul:

```
USE nume_baza_de_date
```

Trebuie să aveți unele privilegii de acces la baza de date; altfel, nu o puteți selecta. De fapt, nu este necesar să selectați o bază de date pentru a folosi tabelele acesteia, deoarece puteți face referire la tabelele sale folosind forma `nume_baza_de_date.nume_tabel`. Totuși, este mult mai convenabil să faceți referire la tabele fără a fi necesar să specificați un calificator pentru baza de date.

Selectarea unei baze de date prestabilite nu înseamnă că aceasta trebuie să fie baza de date prestabilită pe durata conexiunii. Puteți emite orice număr de instrucțiuni `USE` pentru a comuta de la o bază de date la alta oricând doriți, cu condiția să aveți privilegii de

acces pentru a le folosi. De asemenea, selectarea unei baze de date nu vă obligă să folosiți tabele numai din acea bază de date. Puteți continua să faceți referire la tabele din alte baze de date prin specificarea, alături de numele tabelului, a numelui bazei de date.

Când o conexiune cu serverul se încheie, orice noțiune a serverului cu privire la identitatea bazei de date prestabilite dispare. Astfel, dacă vă conectați din nou la server, acesta nu-și amintește baza de date pe care ați selectat-o anterior. De fapt, așa ceva nici nu ar avea sens, dat fiind că MySQL este multi-fir și poate manipula mai multe conexiuni de la un utilizator dat, care se poate conecta și deconecta în orice ordine. În acest mediu, nu este clar care ar putea fi semnificația noțiunii de „bază de date selectată anterior".

Crearea, ștergerea, indexarea și modificarea tabelelor

MySQL vă permite să creați tabele, să le ștergeți și să le modificați structura folosind instrucțiunile `CREATE TABLE`, `DROP TABLE` și `ALTER TABLE`. Pentru fiecare din aceste instrucțiuni există extensii specifice sistemului MySQL care le măresc utilitatea. Instrucțiunile `CREATE INDEX` și `DROP INDEX` vă permit să adăugați sau să eliminați indexuri din tabelele existente.

Instrucțiunea CREATE TABLE

Tabelele sunt create cu ajutorul instrucțiunii `CREATE TABLE`. Sintaxa completă a acestei instrucțiuni este destul de stranie, datorită numărului foarte mare de clauze opționale, dar în practică instrucțiunea respectivă este, de obicei, relativ simplu de utilizat. De exemplu, toate instrucțiunile `CREATE TABLE` pe care le-am folosit în capitolul 1 sunt prezentate într-o formă rezonabil de simplă.

Ca o ironie, o bună parte din această complexitate suplimentară este determinată de clauzele pe care MySQL le analizează și apoi le aruncă! Vă puteți convinge de dimensiunea acestei complexități consultând Anexa D. Examinați informațiile aferente instrucțiunii `CREATE TABLE` și observați cum o bună parte din sintaxa instrucțiunii este dedicată clauzelor `REFERENCES`, `CONSTRAINT` și `CHECK`. Aceste clauze se referă la chei externe, integritate referențială și restricții privind valorile de intrare. MySQL nu acceptă aceste caracteristici, dar analizează sintaxa pentru a face definițiile tabelelor mai ușor de utilizat decât cele pe care le-ați creat în alte sisteme de baze de date. (Puteți folosi mai ușor acel program cu mai puține editări.) Dacă vă scrieți propriile descrieri de tabel pornind de la zero, puteți uita complet de necesitatea de a manipula aceste clauze. Eu unul nu voi mai sufla un cuvânt despre ele în această secțiune.

Instrucțiunea `CREATE TABLE` precizează cel puțin numele tabelului și o listă a coloanelor din cadrul acestuia. De exemplu

```
CREATE TABLE tabelul_meu
(
    nume CHAR(20),
    varsta INT NOT NULL,
    greutate INT,
    sex ENUM('F','M')
)
```


În afară de coloanele care alcătuiesc un tabel, puteți specifica modul de indexare a tabelului atunci când îl creați. O altă opțiune este de a lăsa tabelul neindexat atunci când îl creați și să adăugați indexuri ulterior. Aceasta este o strategie indicată dacă intenționați să populați tabelul cu foarte multe date înainte de a începe să-l utilizați pentru interogări. Actualizarea indexurilor la inserția fiecărui rând este mult mai lentă decât încărcarea datelor într-un tabel neindexat și crearea indexurilor după inserția tuturor datelor.

Am discutat deja despre sintaxa elementară a instrucțiunii `CREATE TABLE` în capitolul 1, iar în capitolul 2, „Lucrul cu date în MySQL", am discutat modul de descriere a tipurilor de coloane. Presupun că ați citit aceste capitole, deci nu vom repeta aici cele discutate anterior. În schimb, în continuare, vom discuta despre unele extensii importante ale instrucțiunii `CREATE TABLE`, care au fost introduse în MySQL 3.23 și care vă oferă o mare flexibilitate în ceea ce privește modul de construcție a tabelelor:

- Specificatori ai tipului de stocare a tabelului
- Crearea unui tabel numai dacă acesta nu există deja
- Tabele temporare, care sunt șterse automat la încheierea sesiunii client
- Posibilitatea de creare a unui tabel prin simpla selectare a datelor pe care doriți să le conțină

Specificatori ai tipului de stocare a tabelului

Anterior versiunii MySQL 3.23, toate tabelele create de utilizator foloseau metoda de stocare ISAM. În MySQL 3.23, puteți crea în mod explicit tabele în oricare din cele trei tipuri, prin specificarea opțiunii `TYPE = tip` după acea parte a instrucțiunii `CREATE TABLE` care conține lista coloanelor. `Tip` poate avea valorile `MYISAM`, `ISAM` sau `HEAP`. De exemplu:

```
CREATE TABLE tabelul_meu (i INT, c CHAR(20)) TYPE = HEAP
```

Puteți converti tabele dintr-un tip în altul folosind instrucțiunea `ALTER TABLE`:

```
ALTER TABLE tabelul_meu TYPE = ISAM
ALTER TABLE tabelul_meu TYPE = MYISAM
ALTER TABLE tabelul_meu TYPE = HEAP
```

Totuși, nu se recomandă conversia unui tabel la tipul HEAP dacă doriți ca tabelul să existe și după închiderea serverului. Tabelele HEAP sunt stocate în memorie și dispar când serverul închide conexiunea.

Caracteristicile generale ale acestor trei tipuri de tabele sunt următoarele:

- **Tabele MyISAM.** Formatul de stocare MyISAM este tipul de tabel prestabilit în MySQL, începând de la versiunea 3.23:
  - Fișierele pot fi mai mari decât în cazul metodei de stocare ISAM, dacă sistemul de operare însuși permite fișiere mai mari.
  - Datele sunt stocate într-un format independent de mașină, cu bitul cel mai puțin semnificativ scris primul. Aceasta înseamnă că puteți copia tabele de la un calculator la altul, chiar dacă acestea au arhitecturi diferite.
  - Valorile unui index numeric ocupă mai puțin spațiu de stocare, deoarece sunt stocate cu bitul cel mai semnificativ scris primul. Valorile unui index tind să varieze mai rapid în domeniul biților mai puțin semnificativi, deci biții mai semnificativi sunt mai expuși la comprimare.

- Manipularea atributului AUTO_INCREMENT este mai bună decât în cazul tabelelor ISAM. Detaliile despre acest atribut sunt discutate în capitolul 2, în secțiunea „Lucrul cu secvențe".
- Numeroase restricții privind indexarea au fost eliminate. De exemplu, puteți indexa coloane care conțin valori NULL, după cum puteți indexa și coloane de tip BLOB și TEXT.
- Pentru o verificare mai bună a integrității tabelului, fiecare tabel are un indicator care este activat (setat) când tabelul este verificat de către myisamchk. Puteți folosi opțiunea myisamchk --fast pentru a nu efectua verificările tabelelor care nu au fost modificate de la ultima verificare, ceea ce mărește viteza acestei operații administrative. De asemenea, tabelele dispun de un indicator care arată dacă un tabel a fost închis în mod adecvat. Dacă serverul se închide într-o manieră anormală sau calculatorul suferă o cădere, indicatorul se poate folosi pentru a detecta tabelele care trebuie verificate în momentul pornirii serverului.

- **Tabele ISAM.** Formatul de stocare ISAM este formatul mai vechi, folosit înainte de versiunea MySQL 3.23, dar continuă să existe și în prezent. În general, tabelele MyISAM sunt preferate tabelelor ISAM, deoarece prezintă mai puține restricții. Probabil că suportul pentru tabelele ISAM va dispărea, deoarece acest format de stocare este înlocuit de formatul de tabel MyISAM.
- **Tabele HEAP.** Formatul de stocare HEAP creează tabele stocate în memorie, care folosesc rânduri de lungime fixă, ceea ce le face foarte rapide. De asemenea, înseamnă că sunt temporare, în sensul că dispar la terminarea conexiunii de către server. Totuși, spre deosebire de tabelele temporare create cu CREATE TEMPORARY TABLE, tabelele HEAP sunt vizibile și altor clienți. Tabelelor HEAP li se aplică restricții care nu sunt valabile pentru tabelele ISAM sau MyISAM:
  - Indexurile sunt folosite numai pentru comparații care utilizează operatorii = și <=>.
  - O coloană indexată nu poate conține valori NULL.
  - Nu se pot utiliza coloanele BLOB și TEXT.
  - Nu se pot utiliza coloanele AUTO_INCREMENT.

Crearea condiționată a tabelelor

Pentru a crea un tabel numai dacă acesta nu există deja, folosiți instrucțiunea CREATE TABLE IF NOT EXISTS. Puteți folosi această instrucțiune într-o aplicație care nu face nici un fel de presupuneri cu privire la faptul dacă tabelele de care are nevoie au fost sau nu configurate în prealabil și pur și simplu încearcă să creeze tabelele ca un lucru de la sine înțeles. Modificatorul IF NOT EXISTS este util mai ales pentru scripturi rulate ca sarcini de grup cu mysql. În acest context, o instrucțiune CREATE TABLE obișnuită nu funcționează foarte bine. La prima rulare a sarcinii, tabelele sunt create, dar a doua oară se va produce o eroare, deoarece tabelele există deja. Dacă folosiți IF NOT EXISTS, nu este nici o problemă. La prima rulare, tabelele vor fi create, ca în situația anterioară. A doua oară și la rulările ulterioare, încercările de creare a tabelului eșuează, dar nu este generată nici o eroare. Astfel, prelucrarea sarcinii poate continua ca și cum încercarea ar fi reușit.

Tabele temporare

Puteți folosi CREATE TEMPORARY TABLE pentru a crea tabele temporare, care dispar automat la încheierea sesiunii. Aceasta este o facilitate convenabilă, deoarece nu trebuie să mai emiteți o instrucțiune DROP TABLE explicită pentru a scăpa de tabel, iar tabelul nu mai continuă să existe dacă sesiunea se încheie în mod anormal. De exemplu, dacă aveți o interogare încapsulată într-un fișier, pe care o rulați cu mysql și decideți să nu mai așteptați încheierea acesteia, puteți lichida scriptul în perfectă siguranță chiar în timpul execuției, iar serverul va elimina toate tabelele temporare.

În versiunile mai vechi ale sistemului MySQL, nu există tabele temporare autentice, cu excepția situației când dumneavoastră le considerați ca atare. Pentru aplicații care necesită un atare tabel, trebuie să vă amintiți să-l ștergeți personal. Dacă uitați să-l ștergeți, sau dacă la client se produce o eroare care îl obligă să-și încheie prematur execuția, tabelul temporar persistă fără rost, până când cineva îl observă și-l elimină.

Un tabel temporar este vizibil numai clientului care creează tabelul. Numele poate fi același cu al unui tabel permanent existent. Aceasta nu reprezintă o eroare și nici nu constituie un atac la adresa tabelului permanent. Să presupunem că ați creat un tabel temporar în baza de date samp_db, denumit membru. Tabelul membru original devine ascuns (inaccesibil), iar referințele la tabelul membru trimit la tabelul temporar. Dacă emiteți o instrucțiune DROP TABLE membru, tabelul temporar este eliminat și tabelul membru original „reapare". Dacă pur și simplu vă deconectați de la server fără a elimina tabelul temporar, serverul va șterge automat tabelul temporar. La următoarea conectare, tabelul membru original este din nou vizibil.

Mecanismul de ascundere a numelui este valabil la un singur nivel. Cu alte cuvinte, nu puteți crea două tabele temporare cu același nume.

Crearea tabelelor pe baza rezultatelor instrucțiunii SELECT

Unul dintre conceptele cheie ale bazelor de date relaționale este că toate informațiile sunt reprezentate sub forma unui tabel compus din rânduri și coloane, iar rezultatul fiecărei instrucțiuni SELECT este de asemenea un tabel compus din rânduri și coloane. În multe situații, „tabelul" care rezultă dintr-o instrucțiune SELECT este pur și simplu o imagine a unor rânduri și coloane care se derulează de sus în jos pe ecranul dumneavoastră în timp ce vă continuați lucrul. Anterior versiunii MySQL 3.23, dacă doreați să salvați rezultatele unei instrucțiuni SELECT într-un tabel ce urma a fi folosit în interogări viitoare, erau necesare unele operațiuni speciale:

1. Rulați o interogare DESCRIBE sau SHOW COLUMNS pentru a determina tipurile coloanelor din tabelele de unde doriți să capturați informația.
2. Creați un tabel, specificând în mod explicit numele și tipurile coloanelor în care tocmai ați căutat.
3. După crearea tabelului, emiteți o interogare INSERT ... SELECT, pentru a regăsi rezultatele și pentru a le insera în tabel.

În MySQL 3.23, situația s-a schimbat. Instrucțiunea CREATE TABLE ... SELECT elimină toate acele încurcături dizgrațioase și permite crearea instantanee a unui nou tabel folosind rezultatul unei interogări SELECT arbitrare. Puteți face această operație într-o

singură etapă, fără a fi obligat să cunoaşteţi sau să specificaţi tipurile de date ale coloanelor pe care le regăsiţi. Astfel, este excepţional de simplu să creaţi un tabel complet populat cu datele care vă interesează şi gata de a fi utilizat în interogări ulterioare.

Puteţi copia un tabel prin selectarea integrală a conţinutului său (fără clauza WHERE) sau puteţi crea o copie vidă prin adăugarea unei clauze WHERE care eşuează întotdeauna:

```
CREATE TABLE nume_tbl_nou SELECT * FROM nume_tbl
CREATE TABLE nume_tbl_nou SELECT * FROM nume_tbl WHERE 1 = 0
```

Crearea unei copii vide este utilă dacă doriţi să încărcaţi un fişier de date în fişierul original folosind LOAD DATA, dar nu sunteţi sigur dacă aveţi opţiunile pentru specificarea corectă a formatului datelor. Nu doriţi să vă treziţi cu înregistrări deteriorate în tabelul original dacă nu precizaţi corect opţiunile din prima încercare! Utilizarea unei copii vide a tabelului original vă permite să exersaţi opţiunile LOAD DATA pentru specificarea delimitatorilor de coloană şi de linie, până când datele dumneavoastră de intrare sunt interpretate corect. După ce aţi realizat acest lucru, puteţi încărca datele în tabelul original.

Puteţi combina instrucţiunea CREATE TEMPORARY TABLE cu SELECT pentru a crea un tabel temporar ca o copie a lui însuşi:

```
CREATE TEMPORARY TABLE tabelul_meu SELECT * FROM tabelul_meu
```

Această instrucţiune vă permite să modificaţi conţinutul tabelului tabelul_meu fără a afecta originalul. Acest fapt este util când doriţi să încercaţi unele interogări care aduc modificări conţinutului tabelului, fără a modifica tabelul original. Pentru a folosi scripturi scrise anterior care folosesc numele tabelului original, nu trebuie să le editaţi pentru a face referire la un alt tabel; pur şi simplu adăugaţi instrucţiunea CREATE TEMPORARY TABLE la începutul scriptului. Scriptul va crea o copie temporară şi va lucra cu aceasta, iar serverul va şterge copia atunci când scriptul îşi încheie execuţia.

Pentru a crea un tabel sub forma unei copii vide a lui însuşi, folosiţi o clauză WHERE 0 în conjuncţie cu CREATE TEMPORARY ... SELECT:

```
CREATE TEMPORARY TABLE tabel SELECT FROM tabel WHERE 1 = 0
```

Totuşi, există numeroase capcane de care trebuie să vă feriţi când creaţi tabele instantaneu. Când creaţi un tabel prin selectarea unor date în acesta, numele coloanelor sunt preluate de la coloanele pe care le selectaţi. Dacă o coloană este calculată ca rezultat al unei expresii, „numele" coloanei este textul expresiei. O expresie nu este permisă ca nume de coloană. Puteţi constata acest lucru dacă rulaţi următoarea interogare în mysql:

```
mysql> CREATE TABLE tabelul_meu SELECT 1;
ERROR 1166 at line 1: Incorrect column name '1'
```

(Eroare 1166 la linia 1: Nume de coloană incorect '1')

Pentru a remedia eroarea, furnizaţi un alias al coloanei, pentru a atribui coloanei un nume permis:

```
mysql> CREATE TABLE tabelul_meu SELECT 1 AS coloana_mea;
Query OK, 1 row affected (0.01 sec)
```

O problemă asemănătoare apare dacă selectaţi coloane cu acelaşi nume din tabele diferite. Să presupunem că tabelele t1 şi t2 au ambele o coloană c şi dumneavoastră doriţi să creaţi un tabel din toate combinaţiile de rânduri din cele două tabele. Puteţi furniza aliasuri pentru a specifica nume unice de coloană în noul tabel:


```
CREATE TABLE t3 SELECT t1.c AS c1, t2.c AS c2 FROM t1, t2;
```

Crearea unui tabel prin selectarea unor date în acesta nu copiază în mod automat nici un index din tabelul original.

Instrucţiunea DROP TABLE

Ştergerea unui tabel este cu mult mai simplă decât crearea sa, deoarece nu trebuie să specificaţi nimic despre conţinutul tabelului; pur şi simplu trebuie să denumiţi tabelul:

```
DROP TABLE nume_tabel
```

MySQL extinde instrucţiunea DROP TABLE în câteva moduri utile. Mai întâi, puteţi şterge mai multe tabele specificându-le pe toate în aceeaşi instrucţiune:

```
DROP TABLE nume_tabel1, nume_tabel2, ...
```

În al doilea rând, dacă nu sunteţi sigur cu privire la existenţa sau nu a unui tabel, dar doriţi să-l ştergeţi dacă există, puteţi adăuga la instrucţiune opţiunea IF EXISTS. Aceasta determină sistemul MySQL să nu facă reclamaţii şi să nu emită erori dacă tabelul sau tabelele specificate în instrucţiune nu există:

```
DROP TABLE IF EXISTS nume_tabel
```

IF EXISTS este util în scripturi pe care le folosiţi cu clientul mysql, deoarece mysql, în mod prestabilit, îşi încheie execuţia la producerea unei erori, iar încercarea de a elimina un tabel care nu există constituie o eroare. De exemplu, puteţi avea un script de configurare care creează tabele pe care le folosiţi ca bază pentru prelucrări ulterioare în alte scripturi. În această situaţie, doriţi să vă asiguraţi că scriptul de configurare porneşte „pe curat". Dacă folosiţi o instrucţiune DROP TABLE normală la începutul scriptului, acesta va eşua prima dată, deoarece tabelele nu au fost niciodată create.

Crearea şi ştergerea indexurilor

Indexurile reprezintă metoda principală de accelerare a accesului la conţinutul tabelelor dumneavoastră, mai ales pentru interogări care implică uniri ale mai multor tabele. Acesta este un subiect suficient de important pentru ca în capitolul 4, „Optimizarea interogărilor", să se discute despre motivele utilizării indexurilor, modul de funcţionare a acestora şi despre modalitatea de utilizare optimă a indexurilor în vederea optimizării interogărilor. În această secţiune, vom discuta despre caracteristicile indexurilor şi despre sintaxa pe care o folosiţi pentru crearea şi ştergerea lor.

Caracteristicile indexurilor

MySQL asigură o flexibilitate remarcabilă în ceea ce priveşte modul de construire a indexurilor. Puteţi indexa coloane individuale sau combinaţii de coloane. De asemenea, puteţi avea mai multe indexuri într-un tabel, dacă doriţi să puteţi căuta rapid o valoare în diferite coloane ale tabelului. Dacă o coloană este de un tip şir diferit de ENUM sau SET, puteţi opta pentru indexarea numai a primelor n caractere (numărate de la stânga la dreapta) din coloană. În cazul în care coloana este predominant unică sub aspectul primelor n caractere, de regulă nu veţi sacrifica performanţele, ba chiar le puteţi îmbunătăţi: indexarea prefixului unei coloane, şi nu a întregii coloane, poate reduce atât dimensiunea indexului, cât şi timpul de acces.

Există şi unele restricţii privind crearea unui index, deşi acestea tind să se „dilueze" pe măsură ce sistemul MySQL continuă să se dezvolte. Tabelul următor prezintă unele diferenţe dintre tabelele ISAM şi MyISAM din punct de vedere al posibilităţilor de indexare:

Caracteristica indexului	Tabele ISAM	Tabele MyISAM
Valori NULL	Interzise	Permise
Coloane BLOB şi TEXT	Nu pot fi indexate	Pot fi indexate
Indexuri per tabel	16	32
Coloane per index	16	16
Dimensiune maximă a rândului index	256 octeţi	500 octeţi

Din tabel se poate observa că, în tabelele ISAM, o coloană indexată trebuie declarată NOT NULL, iar coloanele BLOB şi TEXT nu pot fi indexate. Tipul de tabel MyISAM elimină aceste restricţii legate de crearea indexului şi le atenuează pe altele. O consecinţă a acestor diferenţe între cele două tipuri de tabele în ceea ce priveşte caracteristicile indexului este aceea că, în funcţie de versiunea dumneavoastră de MySQL, s-ar putea să nu fiţi capabil de a indexa anumite coloane. De exemplu, puteţi folosi numai tabele ISAM dacă versiunea dumneavoastră de MySQL este anterioară versiunii 3.23, ceea ce înseamnă că nu puteţi indexa anumite coloane dacă doriţi ca acestea să poată conţine valori NULL.

Dacă dispuneţi de versiunea MySQL 3.23 sau ulterioară, puteţi avea tabele mai vechi, care au fost create de la bun început ca tabele ISAM. Le puteţi converti cu uşurinţă în formatul de stocare MyISAM folosind instrucţiunea ALTER TABLE, care vă permite să beneficiaţi de unele dintre caracteristicile de indexare mai recente:

```
ALTER TABLE nume_tabel TYPE = MYISAM
```

Crearea indexurilor

Puteţi crea indexuri pentru un tabel nou atunci când folosiţi CREATE TABLE sau puteţi adăuga indexuri la tabelele existente utilizând CREATE INDEX sau ALTER TABLE. Instrucţiunea CREATE INDEX a fost introdusă în MySQL 3.22, dar puteţi folosi ALTER TABLE dacă versiunea dumneavoastră de MySQL este anterioară versiunii respective. (În prezent, MySQL mapează intern instrucţiunea CREATE INDEX peste ALTER TABLE.)

Puteţi specifica dacă indexul poate conţine sau nu valori duplicate. Dacă indexul nu poate conţine asemenea valori, atunci trebuie creat ca index de tip PRIMARY KEY sau UNIQUE. Pentru un index unic pe o singură coloană, această caracterizare asigură imposibilitatea existenţei valorilor duplicate în coloana respectivă. Pentru un index unic pe mai multe coloane, este astfel garantată imposibilitatea duplicării unei combinaţii de valori.

Un index PRIMARY KEY este foarte asemănător cu un index UNIQUE. De fapt, indexul PRIMARY KEY nu este altceva decât un index UNIQUE cu numele PRIMARY. Aceasta înseamnă că un tabel poate conţine un singur index PRIMARY KEY, deoarece nu puteţi avea două indexuri cu acelaşi nume. Puteţi plasa mai multe indexuri UNIQUE într-un tabel, deşi nu se prea obişnuieşte.

Pentru a adăuga un index la un tabel existent, puteţi folosi ALTER TABLE sau CREATE INDEX. Instrucţiunea ALTER TABLE cea este mai versatilă dintre cele două, deoarece o puteţi folosi pentru a crea un index obişnuit, un index UNIQUE sau un index PRIMARY KEY:


```
ALTER TABLE nume_tbl ADD INDEX nume_index (lista_col)
ALTER TABLE nume_tbl ADD UNIQUE nume_index (lista_col)
ALTER TABLE nume_tbl ADD PRIMARY KEY (lista_col)
```

nume_tbl este numele tabelului la care se va adăuga indexul, iar *lista_col* precizează coloana sau coloanele care trebuie indexate. Dacă indexul este alcătuit din cel puţin două coloane, numele acestora vor fi separate prin virgulă. Numele indexului *nume_index* este opţional, deci îl puteţi omite, iar MySQL va selecta un nume în funcţie de numele primei coloane indexate. Instrucţiunea ALTER TABLE vă permite să specificaţi mai multe modificări ale tabelului într-o singură instrucţiune, deci puteţi crea mai multe indexuri în acelaşi timp.

Instrucţiunea CREATE INDEX poate adăuga într-un tabel un index obişnuit sau UNIQUE:

```
CREATE UNIQUE INDEX nume_index ON nume_tbl (lista_col)
CREATE INDEX nume_index ON nume_tbl (lista_col)
```

Opţiunile nume_tbl, nume_index şi lista_col au aceeaşi semnificaţie ca în cazul instrucţiunii ALTER TABLE. Totuşi, numele indexului nu este opţional. Nu puteţi crea un index PRIMARY KEY cu instrucţiunea CREATE INDEX.

Pentru a crea un index într-un tabel nou atunci când emiteţi o instrucţiune CREATE TABLE, folosiţi o sintaxă similară instrucţiunii ALTER TABLE, dar specificaţi clauzele de creare a indexului în acea parte a instrucţiunii unde declaraţi coloanele tabelului:

```
CREATE TABEL nume_tabel
(
    ...
    INDEX nume_index (lista_coloane),
    UNIQUE nume_index (lista_coloane),
    PRIMARY KEY (lista_coloane),
    ...
)
```

În ceea ce priveşte ALTER TABLE, numele indexului este opţional pentru INDEX şi UNIQUE, iar MySQL va alege un nume, dacă dumneavoastră nu o faceţi.

Ca un caz special, puteţi crea un index PRIMARY KEY cu o singură coloană prin adăugarea opţiunii PRIMARY KEY la sfârşitul declaraţiei coloanei:

```
CREATE TABLE tabelul_meu
(
    i INT NOT NULL PRIMARY KEY
)
```

Instrucţiunea de mai sus este echivalentă cu următoarea:

```
CREATE TABLE tabelul_meu
(
    i INT NOT NULL,
    PRIMARY KEY (i)
)
```

Fiecare din exemplele anterioare de creare a unui tabel au specificat NOT NULL pentru coloanele indexate. Dacă aveți tabele ISAM, aceasta este o cerință obligatorie, deoarece nu puteți indexa coloane care pot conține valori NULL. Dacă aveți tabele MyISAM, coloanele indexate pot conține și valoarea NULL, cu condiția ca indexul să nu fie de tip PRIMARY KEY.

Dacă indexați un prefix al unei coloane șir (primele n caractere, de la stânga la dreapta, ale valorilor coloanei), sintaxa pentru denumirea coloanei într-un specificator lista_coloane este nume_coloana(n), nu doar un simplu nume_coloana. De exemplu, prima din următoarele instrucțiuni creează un tabel cu două coloane CHAR și un index care folosește ambele coloane. Cea de-a doua instrucțiune este similară, dar indexează un prefix pentru fiecare coloană:

```
CREATE TABLE tabelul_meu
(
    nume CHAR(30),
    adresa CHAR(60),
    INDEX (nume,adresa)
)
CREATE TABLE tabelul_meu
(
    nume CHAR(30),
    adresa CHAR(60),
    INDEX (nume(10),adresa(20))
)
```

În unele circumstanțe, vi se poate părea necesară indexarea unui prefix de coloană. De exemplu, lungimea rândurilor index are o limită superioară, deci puteți fi obligat să folosiți prefixe dacă lungimea coloanelor indexate ar fi depășit limita respectivă. Prefixele sunt de asemenea necesare pentru coloanele BLOB sau TEXT din indexurile tabelelor MyISAM.

Indexarea unui prefix de coloană impune modificări pe care le puteți efectua în coloană la o dată ulterioară; nu puteți reduce lungimea coloanei până la o dimensiune mai mică decât lungimea prefixului fără a șterge indexul și a-l re-crea ulterior folosind un prefix mai scurt.

Ștergerea indexurilor

Puteți șterge indexuri folosind DROP INDEX sau ALTER TABLE. Similar instrucțiunii CREATE INDEX, instrucțiunea DROP INDEX este manipulată intern ca o instrucțiune ALTER TABLE și a fost introdusă în MySQL versiunea 3.22. Sintaxa instrucțiunii de ștergere a indexului este următoarea:

```
DROP INDEX nume_index ON nume_tabel
ALTER TABLE nume_tabel DROP INDEX nume_index
ALTER TABLE nume_tabel DROP PRIMARY KEY
```

Primele două instrucțiuni sunt echivalente. Cea de-a treia este folosită numai pentru ștergerea unui index de tip PRIMARY KEY; în acest caz, nu este necesar nici un nume de index, deoarece un tabel poate conține o singură asemenea cheie. Dacă nici un index nu a fost creat în mod explicit ca PRIMARY KEY, dar tabelul are două sau mai multe indexuri de tip UNIQUE, MySQL îl șterge pe primul dintre acestea.

Indexurile pot fi afectate dacă ștergeți coloane dintr-un tabel. Dacă ștergeți o coloană care face parte dintr-un index, coloana va fi eliminată și din index. Dacă sunt șterse toate coloanele care alcătuiesc un index, atunci întregul index va fi șters.

Instrucțiunea ALTER TABLE

ALTER TABLE este o instrucțiune versatilă în MySQL și se poate folosi în multe scopuri. Deja am văzut câteva dintre posibilitățile sale (pentru crearea și ștergerea indexurilor și pentru conversia tabelelor dintr-un format de stocare în altul). În această secțiune, vom discuta despre alte câteva „aptitudini" ale acestei instrucțiuni. Sintaxa completă a instrucțiunii ALTER TABLE este descrisă în Anexa D.

ALTER TABLE este utilă atunci când sesizați că structura unui tabel nu mai reflectă finalitatea acestuia. Puteți folosi tabelul pentru a înregistra informații suplimentare, sau poate că tabelul conține informații care au devenit superflue. Poate că dimensiunea coloanelor existente este prea mică sau poate că le-ați declarat mai mari decât este necesar și doriți să le micșorați pentru a economisi spațiu și pentru a îmbunătăți performanțele interogărilor. Sau poate că pur și simplu ați tastat incorect numele tabelului atunci când ați emis instrucțiunea CREATE TABLE. Iată câteva exemple:

- Efectuați un chestionar bazat pe Web și stocați rezultatele fiecărui chestionar completat sub forma unei înregistrări într-un tabel. Apoi, decideți să modificați chestionarul pentru a adăuga unele întrebări suplimentare. Pentru a face loc noilor întrebări, trebuie să mai adăugați coloane în tabel.
- Derulați un proiect de cercetare. Repartizați numere de caz înregistrărilor de cercetare folosind o coloană AUTO_INCREMENT. Nu v-ați așteptat ca finanțarea proiectului să dureze suficient pentru a permite generarea a mai mult de aproximativ 50000 de înregistrări, deci declarați coloana de tip UNSIGNED SMALLINT, care poate conține maximum 65536 valori unice. Totuși, finanțarea proiectului a fost reluată și se pare că veți mai putea genera încă 50000 de înregistrări. Trebuie ca tipul coloanei să fie mai mare, pentru a permite inserția mai multor numere de caz.
- Modificările dimensionale pot avea loc și în sens invers. Poate că ați creat o coloană CHAR(255), dar acum vă dați seama că nici o valoare din acest tabel nu are o lungime mai mare de 100 caractere. Puteți reduce dimensiunea coloanei pentru a economisi spațiu.

Sintaxa instrucțiunii ALTER TABLE se prezintă astfel:

```
ALTER TABLE nume_tabel actiune, ...
```

Fiecare parametru actiune specifică o modificare pe care doriți să o efectuați în tabel. MySQL extinde instrucțiunea ALTER TABLE, permițându-vă să specificați mai multe acțiuni, separate prin virgulă. Acest lucru este util pentru reducerea volumului de text introdus de la tastatură, dar un motiv mai important al introducerii acestei extensii îl constituie imposibilitatea de a transforma tabelele cu rânduri de lungime variabilă în tabele cu rânduri de lungime fixă fără a fi obligat să transformați în același timp toate coloanele VARCHAR în coloane CHAR.

Exemplele următoare prezintă câteva dintre caracteristicile instrucțiunii `ALTER TABLE`:

- **Redenumirea unui tabel.** O operație simplă; pur și simplu specificați numele vechi și numele nou:

```
ALTER TABLE nume_tabel RENAME AS nume_nou_tabel
```

În MySQL 3.23, care dispune de tabele temporare, redenumirea unui tabel temporar cu un nume care deja există în baza de date determină ascunderea tabelului original pe durata existenței tabelului temporar. Această operație este similară cu procedeul de ascundere a unui tabel prin crearea unui tabel temporar cu același nume.

- **Modificarea tipului unei coloane.** Pentru a modifica tipul unei coloane, puteți folosi una din clauzele `CHANGE` sau `MODIFY`. Să presupunem că în tabelul `tabelul_meu` există o coloană de tip `SMALLINT UNSIGNED` și că doriți să o transformați în `MEDIUMINT UNSIGNED`. Puteți face aceasta folosind oricare din următoarele două comenzi:

```
ALTER TABLE tabelul_meu MODIFY i MEDIUMINT UNSIGNED
ALTER TABLE tabelul_meu CHANGE i i MEDIUMINT UNSIGNED
```

De ce numele coloanei este menționat de două ori în comanda care folosește `CHANGE`? Deoarece `CHANGE` are posibilitatea (pe care `MODIFY` nu o are), în afară de modificarea tipului coloanei, de a redenumi coloana. Dacă ați fi vrut să redenumiți coloana din `i` în `j` simultan cu schimbarea tipului, ați fi procedat astfel:

```
ALTER TABLE tabelul_meu CHANGE i j MEDIUMINT UNSIGNED
```

Aspectul important este că mai întâi denumiți coloana pe care doriți s-o modificați și apoi specificați o declarație completă a coloanei, care include numele coloanei. Numele coloanei trebuie inclus în declarație, chiar dacă este același ca vechiul nume.

Un motiv important pentru modificarea tipurilor de coloană constă în îmbunătățirea eficienței interogărilor pentru uniri care compară coloane din două tabele. O comparație are loc mai rapid atunci când coloanele comparate sunt de același tip. Să presupunem că rulați o interogare ca aceasta:

```
SELECT ... FROM t1, t2 WHERE t1.nume = t2.nume
```

Dacă `t1.nume` este `CHAR(10)` și `t2.nume` este `CHAR(15)`, interogarea nu va rula la fel de rapid ca în cazul în care ambele coloane ar fi fost de tipul `CHAR(15)`. Puteți atribui celor două coloane același tip prin modificarea tipului coloanei `t1.nume`, folosind oricare din următoarele două comenzi:

```
ALTER TABLE t1 MODIFY nume CHAR(15)
ALTER TABLE t1 CHANGE nume nume CHAR(15)
```

Pentru versiuni de MySQL anterioare versiunii 3.23, este esențial ca tipurile coloanelor dintr-o unire să fie identice, în caz contrar indexurile neputând fi folosite pentru comparație. Pentru versiunea 3.23 și versiunile ulterioare, indexurile se pot folosi pentru tipuri diferite, dar interogarea va fi din nou mai rapidă dacă tipurile sunt identice.

- **Conversia unui tabel cu rânduri de lungime variabilă în tabel cu rânduri de lungime fixă.** Să presupunem că aveți un tabel `chartbl` care conține coloane `VARCHAR` pe care doriți să le convertiți în coloane `CHAR`, pentru a vedea care sunt îmbunătățirile de performanță pe care le obțineți. (În general, tabelele cu rânduri de lungime fixă pot fi prelucrate mai rapid decât tabelele cu rânduri de lungime variabilă.) Tabelul a fost creat astfel:

```
CREATE TABLE chartbl (nume VARCHAR(40), adresa VARCHAR(80))
```


În acest caz, problema este că trebuie să modificați simultan toate coloanele, în aceeași instrucțiune `ALTER TABLE`. Nu puteți modifica fiecare coloană în parte, deoarece în acest caz încercarea nu ar avea nici un efect. Dacă rulați interogarea `DESCRIBE chartbl`, veți descoperi că tipul coloanelor este în continuare definir ca `VARCHAR`! Motivul este acela că, dacă modificați o singură coloană la un moment dat, MySQL observă că tabelul conține coloane cu lungime variabilă și reconvertește coloana modificată înapoi la tipul `VARCHAR`, pentru a economisi spațiu. Pentru a rezolva problema, modificați simultan toate coloanele `VARCHAR`:

```
ALTER TABLE chartbl MODIFY nume CHAR(40), MODIFY adresa CHAR(80)
```

În acest moment, instrucțiunea `DESCRIBE` va demonstra că tabelul conține coloane `CHAR`. Acest tip de operații reprezintă un motiv pentru care este important ca instrucțiunea `ALTER TABLE` să accepte mai multe acțiuni în cadrul unei singure instrucțiuni.

Există o capcană pe care trebuie să o cunoașteți atunci când doriți să convertiți un tabel ca acesta: prezența în tabel a coloanelor `BLOB` și `TEXT` va anihila orice încercare de a converti un tabel la un format cu rânduri de lungime fixă. Prezența fie și a unei singure coloane cu lungime variabilă într-un tabel determină existența în tabel a rândurilor cu lungime variabilă, iar aceste tipuri de coloane (`BLOB` și `TEXT` – N.T.) nu au nici un echivalent cu rânduri de lungime fixă.

- **Conversia unui tabel cu rânduri de lungime fixă în tabel cu rânduri de lungime variabilă.** Bun, deci `chartbl` este mai rapid dacă include numai rânduri cu lungime fixă, dar ocupă mai mult spațiu decât este necesar, deci decideți să-l reconvertiți înapoi la forma sa originală, pentru a economisi spațiu. Conversia unui tabel în această direcție este cu mult mai simplă. Trebuie să transformați o singură coloană `CHAR` în `VARCHAR` și MySQL va converti în mod automat celelalte coloane `CHAR`. Pentru a converti tabelul `chartbl`, folosiți oricare dintre următoarele instrucțiuni:

```
ALTER TABLE chartbl MODIFY nume VARCHAR(40)
ALTER TABLE chartbl MODIFY adresa VARCHAR (80)
```

- **Conversia unui tip de tabel.** Dacă ați trecut de la o versiune MySQL anterioară versiunii 3.23 la această versiune sau la una ulterioară, puteți avea tabele mai vechi, care au fost create ca tabele ISAM. Dacă doriți să le creați în format MyISAM, procedați astfel:

```
ALTER TABLE nume_tabel TYPE = MYISAM
```

De ce ați proceda astfel? Un motiv, așa cum s-a arătat în secțiunea „Crearea și ștergerea indexurilor“, este acela că formatul de stocare MyISAM dispune de unele caracteristici de indexare inexistente în formatul ISAM, cum ar fi capacitatea de a indexa valori `NULL` și tipurile de coloane `BLOB` și `TEXT`. Un alt motiv este acela că tabelele MyISAM sunt independente de mașină, deci le puteți deplasa și în alte calculatoare prin copierea directă a fișierelor tabel, chiar dacă sistemele de calcul au arhitecturi hardware diferite. Despre aceste aspecte vom discuta mai detaliat în secțiunea „Realizarea copiilor de siguranță a bazelor de date și copierea acestora“ din capitolul 11.

Obţinerea informaţiilor despre baze de date şi tabele

MySQL furnizează numeroase instrucţiuni pentru obţinerea de informaţii privind bazele de date şi tabelele pe care le conţin acestea. Instrucţiunile respective sunt utile pentru urmărirea conţinutului bazelor dumneavoastră de date şi pentru a vă reaminti structura tabelelor. De asemenea, le puteţi folosi ca ajutor în utilizarea instrucţiunii `ALTER TABLE`; este mai uşor a determina modul de specificare a unei modificări într-o coloană atunci când puteţi afla cum este definită coloana la un moment dat.

Instrucţiunea `SHOW` poate fi folosită pentru a se obţine informaţii privind numeroase aspecte ale bazelor de date şi ale tabelelor dumneavoastră:

`SHOW DATABASES`	Afişează bazele de date din server
`SHOW TABLES`	Afişează tabelele din baza de date curentă
`SHOW TABLES FROM` *nume_db*	Afişează tabelele din baza de date specificată
`SHOW COLUMNS FROM` *nume_tabel*	Afişează informaţiile despre coloanele din tabelul specificat
`SHOW INDEX FROM` *nume_tabel*	Afişează informaţiile despre indexurile din tabelul specificat
`SHOW TABLE STATUS`	Afişează informaţii descriptive despre tabelele din baza de date prestabilită
`SHOW TABLE STATUS FROM` *nume_db*	Afişează informaţii descriptive despre tabelele din baza de date specificată

Instrucţiunile `DESCRIBE` *nume_tabel* şi `EXPLAIN` *nume_tabel* sunt sinonime cu instrucţiunea `SHOW COLUMNS FROM` *nume_tabel*.

Comanda `mysqlshow` furnizează o parte din informaţiile oferite de instrucţiunea `SHOW`, ceea ce vă permite să obţineţi din interpretor informaţiile privind bazele de date şi tabelele:

`% mysqlshow`	Afişează bazele de date din server
`% mysqlshow` *nume_db*	Afişează tabelele din baza de date specificată
`% mysqlshow` *nume_db nume_tabel*	Afişează informaţiile despre coloanele din tabelul specificat
`% mysqlshow --keys` *nume_db nume_tabel*	Afişează informaţiile despre indexurile din tabelul specificat
`% mysqlshow --status` *nume_db*	Afişează informaţii descriptive despre tabelele din baza de date specificată

Utilitarul `mysqldump` vă permite să vizualizaţi structura tabelelor dumneavoastră în forma unei instrucţiuni `CREATE TABLE`. (Spre deosebire de instrucţiunea `SHOW COLUMNS`, eu sunt de părere că rezultatele comenzii `mysqldump` sunt mai uşor de citit şi afişează şi indexurile în tabel.) Dar, dacă folosiţi `mysqldump`, nu uitaţi să invocaţi această comandă cu opţiunea `--no-data`, ca să nu vă împotmoliţi în datele tabelului dumneavoastră!

```
% mysqldump --no-data nume_db nume_tabel
```

Atât pentru `mysqlshow` cât şi pentru `mysqldump`, puteţi specifica opţiunile uzuale, cum ar fi `--host`, pentru a vă conecta la un server instalat pe o altă gazdă.

Regăsirea înregistrărilor

Nu are nici un sens să plasaţi înregistrări într-o bază de date dacă nu le regăsiţi la un moment ulterior şi nu le utilizaţi într-un fel sau altul. Acesta este şi rostul instrucţiunii `SELECT`: să vă ajute să ajungeţi la datele dumneavoastră. `SELECT` este probabil instrucţiunea cea mai folosită a limbajului SQL, dar poate fi şi cea mai dificilă; restricţiile pe care le folosiţi pentru a alege rândurile pot fi arbitrar de complexe şi pot necesita, în multe tabele, comparaţii între coloane.

Sintaxa de bază a instrucţiunii `SELECT` se prezintă astfel:

`SELECT` *lista_selectie*	Coloanele care trebuie selectate
`FROM` *lista_tabele*	Tabelele din care sunt selectate rândurile
`WHERE` *restrictie_primara*	Condiţiile pe care trebuie să le satisfacă rândurile
`GROUP BY` *grupare_coloane*	Modul de grupare a rezultatelor
`ORDER BY` *sortare_coloane*	Modul de sortare a rezultatelor
`HAVING` *restrictie_secundara*	Condiţii secundare pe care trebuie să le satisfacă rândurile
`LIMIT` *numar*	Limitare aplicată rezultatelor

Toate elementele din această sintaxă sunt opţionale, cu excepţia cuvântului `SELECT` şi a parametrului `lista_selectie`, care specifică datele pe care doriţi să le regăsiţi. Unele sisteme de baze de date impun şi utilizarea clauzei `FROM`. MySQL nu impune această clauză, ceea ce vă permite să evaluaţi expresii fără a face referire la vreun tabel:

```
SELECT SQRT(POW(3,2)+POW(4,2))
```

În capitolul 1 am acordat o atenţie deosebită instrucţiunii `SELECT`, concentrându-ne cu precădere asupra listei cu selecţia coloanelor şi asupra clauzelor `WHERE`, `GROUP BY`, `ORDER BY`, `HAVING` şi `LIMIT`. În acest capitol, ne vom concentra asupra aspectului instrucţiunii `SELECT` care este, poate, cel mai derutant: unirea. Vom discuta despre tipurile de uniri pe care le acceptă MySQL, despre sensul şi modul de specificare a acestora. Astfel veţi putea întrebuinţa limbajul MySQL într-un mod mai eficient, deoarece, în multe situaţii, adevărata problemă a determinării modului de a scrie o interogare îl constituie sesizarea modalităţii adecvate de unire a tabelelor. De asemenea, poate doriţi să examinaţi secţiunea „Breviar de soluţii", care apare mai târziu în acest capitol. Acolo veţi găsi soluţii la numeroase probleme SQL, din care marea majoritate implică, într-un fel sau altul, instrucţiunea `SELECT`.

O problemă a instrucţiunii `SELECT` este aceea că, atunci când vă întâlniţi prima dată cu un nou tip de problemă, nu este întotdeauna uşor să determinaţi modul de a scrie o instrucţiune `SELECT` pentru a o rezolva. Totuşi, după ce v-aţi dat seama cum trebuie scrisă interogarea, puteţi folosi experienţa acumulată atunci când veţi întâlni probleme similare în viitor. `SELECT` este, probabil, instrucţiunea în cazul căreia experienţa acumulată este cea mai importantă pentru capacitatea dumneavoastră de a o folosi eficient, pur şi simplu datorită imensei varietăţi de moduri în care puteţi utiliza instrucţiunea.

Pe măsură ce dobândiţi experienţă, o veţi putea aplica mai uşor la noile probleme şi vă veţi trezi gândindu-vă astfel: „A, da, asta-i una din chestiile alea cu `LEFT JOIN`", respectiv „Aha, păi asta este o unire pe trei căi limitată de perechile comune de coloane cheie".

(De fapt, mi-e cam jenă să spun asta. Vi se poate părea încurajator să aflați că experiența vă poate fi de folos. Pe de altă parte, e puțin cam înspăimântător să ajungeți să gândiți folosind un astfel de vocabular!)

În următoarele câteva secțiuni, care demonstrează modul de utilizare a formelor de operatori de unire acceptați de MySQL, majoritatea exemplelor folosesc următoarele două tabele. Tabelele sunt mici, ceea ce le face suficient de simple pentru a permite sesizarea imediată a efectului fiecărui tip de unire:

tabelul t1:

i1	c1
1	a
2	b
3	c

tabelul t2:

i2	c2
2	c
3	b
4	a

Unirea normală

Cea mai simplă unire este unirea normală, în care este denumit un singur tabel. În acest caz, rândurile sunt selectate din tabelul specificat:

```
SELECT * FROM t1
```

i1	c1
1	a
2	b
3	c

Unii autori nu consideră această formă a instrucțiunii `SELECT` ca fiind o unire și folosesc acest termen numai pentru instrucțiunile `SELECT` care regăsesc înregistrări din două sau mai multe tabele. Eu cred că este o problemă de perspectivă.

Unirea completă

Dacă sunt specificate mai multe tabele, cu numele separate prin virgulă, se execută o unire completă. De exemplu, dacă uniți două tabele, fiecare rând din primul tabel este combinat cu fiecare rând din al doilea tabel:

```
SELECT t1.*,t2.* FROM t1, t2
```

i1	c1	i2	c2
1	a	2	c
2	b	2	c
3	c	2	c
1	a	3	b
2	b	3	b
3	c	3	b
1	a	4	a
2	b	4	a
3	c	4	a

O unire completă se mai numește și unire încrucișată, deoarece fiecare rând al fiecărui tabel este încrucișat cu fiecare rând din toate celelalte tabele pentru a produce toate combinațiile posibile. Această operație mai este cunoscută și sub numele de produs cartezian. Unirea în acest mod a tabelelor are potențialul de a genera un număr foarte mare de rânduri, deoarece numărul de rânduri posibil este produsul numerelor de rânduri din fiecare tabel. O unire completă între trei tabele care conțin 100, 200 și 300 de rânduri poate returna 100 × 200 × 300 = 6 milioane de rânduri, ceea ce înseamnă un număr foarte mare, chiar dacă tabelele în sine sunt mici. În situații precum aceasta, în mod normal se va folosi o clauză `WHERE` pentru a reduce setul de rezultate la o dimensiune mai rezonabilă!

Dacă adăugați în clauza `WHERE` o condiție care să determine coroborarea tabelelor în funcție de valorile anumitor coloane, unirea devine ceea ce se numește o echi-unire, deoarece selectați numai rândurile cu valori egale din coloanele specificate:

```
SELECT t1.*, t2.* FROM t1, t2 WHERE t1.i1 = t2.i2
```

i1	c1	i2	c2
2	b	2	c
3	c	3	b

Tipurile de unire `JOIN`, `CROSS JOIN` și `INNER JOIN` sunt echivalente cu operatorul de unire ,.

O unire `STRAIGHT_JOIN` este similară unei uniri complete, dar tabelele sunt unite în ordinea precizată în clauza `FROM`. În mod normal, utilitarul de optimizare din MySQL își ia libertatea de a rearanja ordinea tabelelor dintr-o unire completă pentru o regăsire mai rapidă a rândurilor. Ocazional, utilitarul de optimizare nu va efectua o alegere optimă, alegere pe care o puteți impune folosind cuvântul cheie `STRAIGHT_JOIN`.

`STRAIGHT_JOIN` poate fi specificat în două puncte ale instrucțiunii `SELECT`. Îl puteți insera între cuvântul cheie `SELECT` și lista de selecție, pentru a avea un efect global asupra tuturor unirilor complete din instrucțiune, respectiv îl puteți specifica în clauza `FROM`. Următoarele două instrucțiuni sunt echivalente:

```
SELECT STRAIGHT_JOIN ... FROM tabel1, tabel2, tabel3 ...
SELECT ... FROM tabel1 STRAIGHT_JOIN tabel2 STRAIGHT_JOIN tabel3 ...
```

Explicitarea referințelor la coloane

Referințele la coloanele tabelelor dintr-o instrucțiune `SELECT` trebuie să trimită fără echivoc la un singur tabel denumit în clauza `FROM`. Dacă este specificat un singur tabel, nu există nici un echivoc, deoarece toate coloanele trebuie să aparțină tabelului respectiv. Dacă sunt denumite mai multe tabele, toate numele de coloană care apar într-un singur tabel sunt, de asemenea, fără echivoc. Totuși, dacă numele coloanei apare în mai multe tabele, referirile la coloană trebuie să conțină și numele tabelului, folosind sintaxa *nume_tabel.nume_coloana*, pentru a preciza tabelul la care vă referiți. Dacă un tabel denumit *tabelul_meu1* conține coloanele a și b, iar un tabel denumit *tabelul_meu2* conține coloanele b și c, referințele la coloanele a și c sunt fără echivoc, dar referințele la coloana b trebuie completate sub una din formele `tabelul_meu1.b` sau `tabelul_meu2.b`:

```
SELECT a, tabelul_meu1.b, tabelul_meu2.b, c
FROM tabelul_meu1, tabelul_meu2...
```

În unele situații, un element de explicitare de tip nume de tabel nu este suficient pentru clarificarea unei referințe de coloană. De exemplu, dacă folosiți un tabel de mai multe ori într-o interogare, nu este sufi-

cientă explicitarea unui nume de coloană cu numele tabelului. În acest caz, pentru comunicarea intențiilor dumneavoastră sunt utile aliasurile de tabel. Puteți atribui un alias oricărei instanțe a tabelului și să faceți referiri la coloane din instanța respectivă a tabelului sub forma `nume_alias.nume_coloana`. Următoarea interogare unește un tabel cu el însuși, dar atribuie un alias unei instanțe a tabelului, pentru a permite specificarea fără echivoc a referințelor la coloană:

```
SELECT tabelul_meu.col1, m.col2
       FROM tabelul_meu, tabelul_meu AS m
       WHERE tabelul_meu.col1 > m.col1
```

Unirea la stânga

O echi-unire afișează numai rândurile unde în ambele tabele există o corespondență cu anumite criterii de căutare. O unire la stânga afișează de asemenea corespondențele, dar prezintă și rândurile din tabelul din stânga care *nu* au o corespondență în tabelul din dreapta. Toate coloanele selectate din tabelul din dreapta pentru aceste rânduri sunt afișate ca `NULL`. Mecanismul de funcționare este acela că sunt selectate toate rândurile din tabelul din stânga. Dacă există un rând echivalent în tabelul din dreapta, rândul respectiv este selectat. Dacă nu există nici o echivalență, se selectează totuși un rând, dar acesta este un rând „fals", în care toate coloanele au primit valoarea `NULL`. Cu alte cuvinte, o unire la stânga forțează setul de rezultate să conțină un rând pentru fiecare rând al tabelului din stânga, indiferent dacă pentru rândul respectiv există sau nu un echivalent în tabelul din dreapta. Stabilirea corespondenței se efectuează conform coloanelor denumite într-o clauză `ON` sau `USING()`. Puteți folosi opțiunea `ON` indiferent dacă numele coloanelor pe care le uniți sunt identice:

```
SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2
```

i1	c1	i2	c2
1	a	NULL	NULL
2	b	2	c
3	c	3	b

Clauza `USING()` este similară clauzei `ON`, dar numele coloanei sau ale coloanelor unite trebuie să fie aceleași în fiecare tabel. Interogarea următoare unește coloana `tabelul_meu1.b` cu coloana `tabelul_meu2.b`:

```
SELECT tabelul_meu1.*, tabelulmeu2.* FROM tabelul_meu1
LEFT JOIN tabelul_meu2 USING (b)
```

`LEFT JOIN` este utilă mai ales atunci când doriți să găsiți numai acele rânduri ale tabelului din stânga care nu apar în tabelul din dreapta. Pentru aceasta, adăugați clauza `WHERE` care caută rândurile tabelului din dreapta ce au valori `NULL`:

```
SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2
WHERE t2.i2 IS NULL
```

i1	c1	i2	c2
1	a	NULL	NULL

În mod normal, nu veți selecta coloanele cu valoarea `NULL`, deoarece nu prezintă interes. Ceea ce vă interesează sunt coloanele fără echivalent ale tabelului din stânga:

```
SELECT t1.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2
WHERE t2.i2 IS NULL
```

i1	c1
1	a

Un aspect de care trebuie să vă feriți în ceea ce privește `LEFT JOIN` este acela că, în cazul în care coloanele pe care realizați unirea sunt declarate ca fiind `NOT NULL`, rezultatul poate conține rânduri nedorite.

`LEFT JOIN` are câteva sinonime și variante. `LEFT OUTER JOIN` este sinonim cu `LEFT JOIN`. De asemenea, există o notație ODBC pentru `LEFT JOIN` care este acceptată de către MySQL (expresia `oj` are sensul de *unire exterioară* – outer join):

```
{ oj nume_tabel LEFT OUTER JOIN nume_tabel ON expr_unire }
```

`NATURAL LEFT JOIN` este similar cu `LEFT JOIN`; execută o unire la stânga, stabilind corespondențe pentru toate coloanele cu același nume ale tabelelor din stânga, respectiv din dreapta.

Unele sisteme de baze de date au o instrucțiune `RIGHT JOIN` echivalentă; MySQL nu dispune încă de această instrucțiune.

Scrierea comentariilor

MySQL vă permite să intercalați comentarii în liniile dumneavoastră de program SQL. Acest lucru poate fi util pentru documentarea interogărilor pe care le stocați în fișiere. Puteți scrie comentarii în două moduri. Orice text delimitat de un caracter # la stânga și sfârșitul liniei la dreapta este considerat comentariu. Sunt permise și comentariile folosite în C. Cu alte cuvinte, orice text inclus între simbolurile de început, respectiv sfârșit /* și */ este considerat comentariu. Comentariile în C se pot extinde pe mai multe linii:

```
# acesta este un comentariu pe o singura linie
/* si acesta este un comentariu pe o singura linie */
/* acesta, totusi,
   este un comentariu
   pe mai multe linii
*/
```

Începând de la MySQL versiunea 3.23, puteți „ascunde" cuvintele cheie specifice limbajului MySQL în comentarii de tip C, precedând comentariul prin caracterele `/*!` în loc de `/*`. MySQL examinează acest tip special de comentariu și folosește cuvintele cheie, dar alte servere de baze de date le vor ignora, deoarece fac parte din comentariu. Acest lucru vă ajută să scrieți programe care beneficiază de funcțiile specifice sistemului MySQL atunci când sunt executate în MySQL, dar care pot fi folosite și de către alte servere de baze de date, fără modificări. Următoarele două instrucțiuni sunt echivalente pentru servere de baze de date altele decât MySQL, dar MySQL va efectua o operație `INSERT DELAYED` pentru a doua instrucțiune:

```
INSERT INTO absente (elev_id,data) VALUES(13,"1999-09-28")
INSERT /*! DELAYED */ INTO absente (elev_id,data) VALUES(13,"1999-09-28")
```

Începând de la versiunea MySQL 3.23.3, în afară de stilurile de comentariu descrise anterior, puteți iniția un comentariu cu două liniuțe și un spațiu (- -); orice text plasat între aceste liniuțe și sfârșitul liniei este tratat ca un comentariu. Alte sisteme de baze de date folosesc cele două liniuțe pentru a începe un comentariu; MySQL permite acest lucru, dar impune utilizarea spațiului pentru eliminarea oricărei confuzii.

Instrucțiuni cu expresii precum `5- -7` ar putea fi interpretate ca incluzând un comentariu. Este puțin probabil să scrieți o expresie precum `5- - 7`, deci aceasta este o euristică utilă. Totuși, nu este decât o euristică, deci probabil este bine să folosiți unul dintre celelalte stiluri de comentarii, folosind liniuțele duble numai în programe pe care le portați din alte sisteme de baze de date.

Breviar de soluții

Această secțiune seamănă oarecum cu un „corn al abundenței"; indică modul de scriere a interogărilor pentru rezolvarea diferitelor categorii de probleme. În majoritatea cazurilor, acestea sunt soluții la probleme care mi-au parvenit din lista de corespondență. (Mulțumiri persoanelor din listă care au contribuit cu multe din aceste răspunsuri.)

Rescrierea subselecțiilor sub formă de uniri

Începând cu versiunea 3.24, MySQL va dispune de subselecții. Lipsa acestei caracteristici este una dintre cele mai criticate omisiuni din MySQL, dar un lucru de care mulți par să nu-și dea seama este acela că interogările scrise folosind subselecții pot fi frecvent reformulate folosind uniri. De fapt, chiar și când MySQL va fi dotat cu subselecții, este chiar indicat să examinați interogările pe care sunteți înclinat să le scrieți folosindu-le; deseori este mai eficient să folosiți o unire în locul subselecției.

Rescrierea subselecțiilor care selectează valori corespunzătoare unui criteriu de căutare

Iată un exemplu de interogare care conține o subselecție; interogarea selectează punctajele din tabelul `puncte` obținute la toate testele (adică ignoră punctajele obținute la chestionare):

```
SELECT * FROM puncte WHERE eveniment_id
IN (SELECT eveniment_id FROM eveniment WHERE tip = "T")
```

Aceeași interogare se poate scrie fără o subselecție, prin conversia ei într-o simplă unire:

```
SELECT puncte.* FROM puncte, eveniment
WHERE puncte.eveniment_id = eveniment.eveniment_id AND eveniment.tip = "T"
```

Exemplul următor selectează punctajele obținute de elevii de sex feminin:

```
SELECT * FROM puncte
WHERE elev_id IN (SELECT elev_id FROM elev WHERE sex = "F")
```


Această interogare se poate converti într-o unire după cum se va vedea mai jos:

```
SELECT puncte.* FROM puncte, elev
WHERE puncte.elev_id = elev.elev_id AND elev.sex = "F"
```

Se observă existența unui model. Interogările cu subselecție respectă următoarea formă:

```
SELECT * FROM tabel1 WHERE coloana1
IN (SELECT coloana2 FROM tabel2a WHERE coloana2b = valoare)
```

Asemenea interogări pot fi convertite într-o unire folosind această formă:

```
SELECT tabel1.* FROM tabel1, tabel2
WHERE tabel1.coloana1 = tabel2.coloana2a AND tabel2.coloana2b = valoare
```

Rescrierea subselecțiilor care selectează valori care nu corespund unui criteriu de căutare

O altă categorie comună de interogări cu subselecție caută valorile dintr-un tabel care *nu* se regăsesc în alt tabel. Așa cum am văzut mai sus, tipul de problemă „valori care nu există" poate fi soluționat cu ajutorul unei instrucțiuni `LEFT JOIN`. Iată o interogare cu subselecție care testează absența valorilor dintr-un tabel (îi găsește pe elevii cu prezență 100%):

```
SELECT * FROM elev
WHERE elev_id NOT IN (SELECT elev_id FROM absente)
```

Această interogare se poate rescrie folosind o instrucțiune `LEFT JOIN` după cum urmează:

```
SELECT elev.*
FROM elev LEFT JOIN absente ON elev.elev_id = absente.elev_id
WHERE absente.elev_id IS NULL
```

În termeni generali, forma interogării cu subselecție este următoarea:

```
SELECT * FROM tabel1
WHERE coloana1 NOT EXISTS (SELECT coloana2 FROM tabel2)
```

O interogare cu această formă se poate rescrie astfel:

```
SELECT tabel1.*
WHERE tabel1 LEFT JOIN tabel2
ON tabel1.coloana1 = tabel2.coloana2
WHERE tabel2.coloana2 IS NULL
```

Se presupune că declarația coloanei `tabel2.coloana2` conține atributul `NOT NULL`.

Depistarea valorilor inexistente într-un tabel

Deja am văzut în secțiunea „Regăsirea înregistrărilor" că, atunci când doriți să știți care sunt valorile dintr-un tabel inexistente într-un alt tabel, folosiți o instrucțiune `LEFT JOIN` pentru cele două tabele și căutați rândurile în care este selectată valoarea `NULL` din al doilea tabel. Atunci a fost prezentată o situație simplă, care folosea următoarele două tabele:

tabelul t1:

i1	c1
1	a
2	b
3	c

tabelul t2:

i2	c2
2	c
3	b
4	a

Instrucțiunea `LEFT JOIN` pentru găsirea tuturor valorilor din coloana `t1.i1` care nu se află în coloana `t2.i2` are forma următoare:

```
SELECT t1.* FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i2
WHERE t2.i2 IS NULL
```

i1	c1
1	a

Acum, să ne gândim la o versiune mai dificilă a întrebării „Care sunt valorile care lipsesc?". În proiectul de evidență a rezultatelor școlare, menționat pentru prima dată în capitolul 1, avem un tabel `elev` care conține elevii, un tabel `eveniment` care prezintă evenimentele de tip examinare care s-au produs, respectiv un tabel `puncte` care enumeră punctajele obținute de fiecare elev la fiecare eveniment de tip examinare. Totuși, dacă un elev a fost bolnav la data susținerii unui chestionar sau a unui test, tabelul `puncte` nu va conține nici un punctaj al elevului pentru evenimentul respectiv, deci este necesară reluarea chestionarului sau a testului. Cum putem găsi aceste înregistrări lipsă, pentru a ne putea asigura că elevii respectivi susțin repetarea examinării?

Problema constă în determinarea elevilor care nu au nici un punctaj la o examinare, pentru fiecare examinare în parte. Cu alte cuvinte, dorim să aflăm care sunt combinațiile elev-eveniment care nu se regăsesc în tabelul `puncte`. Această formulare de tip „care sunt valorile inexistente" este o sugestie a faptului că se dorește o instrucțiune `LEFT JOIN`. Unirea nu va fi la fel de simplă ca în exemplul anterior, totuși, deoarece nu căutăm numai valori inexistente într-o singură coloană, ci o combinație între două coloane.

Combinațiile pe care le căutăm sunt toate combinațiile elev-eveniment, care se produc din încrucișarea tabelului `elev` cu tabelul `eveniment`:

```
FROM elev, eveniment
```

Apoi, luăm rezultatul unirii respective și executăm o instrucțiune `LEFT JOIN` cu tabelul `puncte` pentru a găsi perechile căutate:

```
FROM elev, eveniment
    LEFT JOIN puncte ON elev.elev_id = puncte.elev_id
                    AND eveniment.eveniment_id = puncte.eveniment_id
```

Rețineți că clauza `ON` permite unirea rândurilor din tabelul `puncte` în conformitate cu valorile echivalente din diferite tabele. Aceasta este cheia pentru rezolvarea problemei. Instrucțiunea `LEFT JOIN` forțează generarea unui rând pentru fiecare rând obținut din unirea încrucișată a tabelelor `elev` și `eveniment`, chiar dacă nu există nici o înregistrare corespunzătoare în tabelul `puncte`. Rândurile din setul de rezultate pentru aceste înregistrări lipsă din tabelul `puncte` pot fi identificate prin faptul că toate coloanele din tabelul `puncte` vor avea valoarea `NULL`. Putem selecta aceste înregistrări în clauza `WHERE`. Se poate utiliza orice coloană din tabelul `puncte`, dar, deoarece căutăm punctajele lipsă, probabil că din punct de vedere conceptual este cel mai clar dacă testăm coloana `puncte`:

```
WHERE puncte.puncte IS NULL
```

Putem pune rezultatele în ordine folosind o clauză `ORDER BY`. Cele mai logice două aranjamente sunt în funcție de elev și de eveniment. Îl voi alege pe primul:

```
ORDER BY elev_elev.id, eveniment.eveniment_id
```


Acum, tot ce avem de făcut este să specificăm coloanele pe care dorim să le vedem ca date de ieșire și am terminat. Iată interogarea finală:

```
SELECT
    elev.nume, elev.elev_id,
    eveniment.data, eveniment.eveniment_id, eveniment.tip
FROM
    elev, eveniment
    LEFT JOIN puncte ON elev.elev_id = puncte.elev_id
                    AND eveniment.eveniment_id = puncte.eveniment_id
WHERE
    puncte.puncte IS NULL
ORDER BY
    elev_elev.id, eveniment.eveniment_id
```

La rularea interogării se produc următoarele rezultate:

nume	elev_id	data	eveniment_id	tip
Megan	1	1999-09-16	4	C
Joseph	2	1999-09-03	1	C
Katie	4	1999-09-23	5	C
Devri	13	1999-09-03	1	C
Devri	13	1999-10-01	6	T
Will	17	1999-09-16	4	C
Avery	20	1999-09-06	2	C
Gregory	23	1999-10-01	6	T
Sarah	24	1999-09-23	5	C
Carter	27	1999-09-16	4	C
Carter	27	1999-09-23	5	C
Gabrielle	29	1999-09-16	4	C
Grace	30	1999-09-23	5	C

Aici este un element subtil. Datele de ieșire afișează identificatorii de elev și de eveniment. Coloana `elev_id` apare atât în tabelul `elev`, cât și în tabelul `puncte`, deci la început s-ar putea să credeți că lista de selecție putea conține fie `elev.elev_id`, fie `puncte.elev_id`. Nu este adevărat, deoarece fundamentul capacității de a găsi înregistrările care ne interesează este acela că toate câmpurile din tabelul `puncte` sunt returnate ca fiind `NULL`. Prin selectarea coloanei `puncte.elev_id` s-ar fi obținut ca date de ieșire numai o coloană de valori `NULL`. Un raționament similar se aplică și în cazul coloanei `eveniment_id`, care apare atât în tabelul `eveniment`, cât și în tabelul `puncte`.

Efectuarea unei operații UNION

Dacă doriți să creați un set de rezultate prin selectarea înregistrărilor din mai multe tabele care au aceeași structură, puteți face acest lucru în unele sisteme de baze de date folosind o anumită categorie de instrucțiuni `UNION`. MySQL nu dispune de o asemenea instrucțiune (cel puțin până la MySQL versiunea 3.24), dar puteți ocoli acest neajuns în câteva moduri. În continuare sunt date două posibile soluții:

- Efectuați mai multe interogări SELECT, câte una pentru fiecare tabel. Această soluție este posibilă dacă nu vă interesează ordinea rândurilor pe care le selectați.
- Selectați rândurile din fiecare tabel într-un tabel temporar de stocare. Apoi selectați conținutul acelui tabel. Acest lucru vă permite să sortați rândurile așa cum doriți. În MySQL 3.23 și versiunile ulterioare, puteți rezolva cu ușurință această problemă permițând serverului să creeze automat tabelul de stocare. De asemenea, puteți atribui acestui tabel un caracter temporar, astfel încât să fie șters automat la terminarea sesiunii dumneavoastră de lucru cu serverul.

În programul următor, vom șterge tabelul în mod explicit, pentru a permite serverului să elibereze resursele asociate acestuia. Aceasta este o idee bună dacă sesiunea client va continua să efectueze și alte interogări. De asemenea, vom folosi un tabel HEAP (stocat în memorie) pentru a obține performanțe sporite.

```
CREATE TEMPORARY TABLE tbl_stoc TYPE=HEAP ... FROM tabel1 WHERE...
INSERT INTO tbl_stoc SELECT ... FROM tabel2 WHERE ...
INSERT INTO tbl_stoc SELECT ... FROM tabel3 WHERE ...
...
SELECT * FROM tbl_stoc ORDER BY ...
DROP TABLE tbl_stoc
```

Pentru versiunile de MySQL anterioare versiunii 3.23, ideea este similară, cu excepția faptului că trebuie să declarați în mod explicit coloanele din tabelul *tbl_stoc*, iar instrucțiunea DROP TABLE de la sfârșit este obligatorie, pentru a împiedica existența tabelului după sfârșitul perioadei de viață a sesiunii client:

```
CREATE TABLE tbl_stoc (coloana1 ..., coloana 2 ..., ...)
SELECT ... FROM tabel1 WHERE ...
INSERT INTO tbl_stoc SELECT ... FROM tabel1 WHERE ...
INSERT INTO tbl_stoc SELECT ... FROM tabel2 WHERE ...
INSERT INTO tbl_stoc SELECT ... FROM tabel3 WHERE ...
SELECT * FROM tbl_stoc ORDER BY ...
DROP TABLE tbl_stoc
```

Adăugarea unei coloane cu numere dintr-o secvență

Dacă folosiți instrucțiunea ALTER TABLE pentru a adăuga o coloană AUTO_INCREMENT, coloana este completată automat cu numere dintr-o secvență. Următorul set de instrucțiuni dintr-o sesiune mysql prezintă principiul de funcționare al acestei operații prin crearea unui tabel, inserția unor date în tabel și apoi adăugarea unei coloane AUTO_INCREMENT:

```
mysql> CREATE TABLE t(c CHAR(10));
mysql> INSERT INTO t VALUES("a"),("b"),("c");
mysql> SELECT * FROM t;
```

c
a
b
c


```
mysql> ALTER TABLE t ADD i INT AUTO_INCREMENT NOT NULL PRIMARY KEY
mysql> SELECT * FROM t;
```

c	i
a	1
b	2
c	3

Secvențierea[1] sau resecvențierea unei coloane existente

Dacă aveți o coloană numerică, o puteți secvenția (sau resecvenția dacă a fost secvențiată anterior, dar ați șters rânduri și doriți să renumerotați valorile pentru ca acestea să fie contigui) astfel:

```
ALTER TABLE t MODIFY i INT NULL
UPDATE t SET i = NULL
ALTER TABLE t MODIFY i INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY
```

Dar o modalitate mai simplă de a efectua această operație este de a șterge pur și simplu coloana și de a o adăuga din nou, sub forma unei coloane AUTO_INCREMENT. Instrucțiunea ALTER TABLE permite specificarea mai multor acțiuni, deci toate aceste operații se pot executa într-o singură instrucțiune:

```
ALTER TABLE t
    DROP i,
    ADD i INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY
```

Sortarea într-o ordine neobișnuită

Să presupunem că aveți un tabel care conține personalul unei instituții sportive, cum ar fi o echipă de fotbal: doriți să sortați datele de ieșire după funcția membrilor personalului, astfel încât acestea să apară într-o anumită ordine, cum ar fi următoarea: antrenor, antrenor secund, portari, fundași, mijlocași, atacanți[2]. Definiți coloana cu tipul ENUM și menționați elementele enumerării în ordinea în care doriți să le vedeți. Rezultatele operațiilor de sortare din coloana respectivă vor apărea în ordinea pe care o specificați.

Configurarea unui tabel contor

În secțiunea „Lucrul cu secvențe" din capitolul 2, am prezentat modul de generare a unei secvențe folosind funcția LAST_INSERT_ID(expr). Exemplul de acolo ilustra modul de configurare a unui contor folosind un tabel cu un singur rând. Exemplul este foarte bun pentru un singur contor, dar, dacă aveți nevoie de mai multe contoare, metoda respectivă duce la o multiplicare inutilă a tabelelor. Să presupunem că aveți un sit Web și doriți

[1] Adică transformarea unei secvențe de numere fără nici o ordine într-o secvență ordonată; prin „resecvențiere" se va înțelege revenirea la ordinea din secvență, ordine alterată de o operație sau alta. – N.T.

[2] Am recurs la „adaptarea" traducerii folosind posturile dintr-o echipă de fotbal european, posturi mult mai familiare cititorului român decât cele din cadrul unei echipe de fotbal american, folosite de autorul ediției originale. – N.T.

să inserați în mai multe pagini contoare de genul „această pagină a fost deschisă de nnn ori". Probabil că nu doriți să configurați un tabel contor separat pentru fiecare pagină care are un contor.

O modalitate de a evita crearea de tabele contor multiple este de a crea un singur tabel cu două coloane. O coloană conține o valoare contor; cealaltă conține numele contorului. Putem folosi în continuare funcția `LAST_INSERT_ID()`, dar vom determina rândul căruia i se aplică această funcție folosind numele contorului. Tabelul se prezintă astfel:

```
CREATE TABLE contor
(
    numar INT UNSIGNED,
    nume VARCHAR(255) NOT NULL PRIMARY KEY
)
```

Numele este un șir, astfel încât putem apela un contor în orice mod dorim, motiv pentru care îl vom transforma în coloană `PRIMARY KEY` pentru a preveni duplicarea numelor. Acest lucru presupune că aplicațiile care folosesc tabelul convin asupra numelor pe care le vor folosi. Pentru contoarele noastre din paginile Web, putem asigura unicitatea numelor contoarelor prin simpla utilizare ca nume de contor al unei pagini a numelui căii de acces la pagina respectivă din cadrul arborelui documentului. De exemplu, configurarea unui contor nou pentru pagina de bază a sitului se realizează astfel:

```
INSERT INTO contor (nume) VALUES("index.html")
```

Această instrucțiune inițializează contorul denumit `"index.html"` cu valoarea zero. Pentru a genera următoarea valoare din secvență, incrementați contorul în rândul adecvat al tabelului, după care regăsiți-l folosind funcția `LAST_INSERT_ID()`:

```
UPDATE contor
    SET numar = LAST_INSERT_ID(numar+1)
    WHERE nume = "index.html"
SELECT LAST_INSERT_ID()
```

O metodă alternativă ar fi incrementarea contorului fără a folosi funcția `LAST_INSERT_ID()`, astfel:

```
UPDATE contor SET numar = numar+1 WHERE nume = "index.html"
SELECT numar FROM contor WHERE nume = "index.html"
```

Totuși, metoda nu funcționează corect dacă un alt client incrementează contorul după ce emiteți instrucțiunea `UPDATE` și înainte de a emite instrucțiunea `SELECT`. Puteți rezolva această problemă delimitând cele două instrucțiuni cu instrucțiunile `LOCK TABLES` și `UNLOCK TABLES`, pentru a bloca alți clienți în timp ce folosiți contorul. Dar metoda `LAST_INSERT_ID()` execută aceeași operație mult mai simplu. Deoarece valoarea sa este specifică clientului, obțineți întotdeauna valoarea pe care ați inserat-o, nu pe cea a altui client, și nu trebuie să complicați programul cu blocări pentru a nu permite accesul altor clienți.

Verificarea existenței tabelului

Uneori este util să putem determina, din interiorul unei aplicații, dacă un tabel dat există sau nu. Pentru aceasta, puteți folosi oricare din următoarele instrucțiuni:


```
SELECT COUNT(*) FROM nume_tabel
SELECT * FROM nume_tabel WHERE 1=0
```

Fiecare din aceste instrucțiuni reușește dacă tabelul există, respectiv eșuează dacă tabelul nu există. Ele reprezintă interogări performante pentru această categorie de test și se execută foarte rapid, deci nu aveți nevoie de prea mult timp pentru a le rula. Această strategie este cea mai indicată pentru programele de aplicație pe care le scrieți, deoarece puteți testa succesul sau eșecul interogării și puteți întreprinde acțiuni în consecință. Metoda nu este deosebit de utilă într-un script de grup pe care îl rulați din `mysql`, deoarece la apariția unei erori nu puteți face nimic altceva decât să terminați programul (sau să ignorați eroarea, dar atunci rularea interogării nu mai are nici un sens).

Caracteristici pe care MySQL nu le acceptă

Această secțiune descrie caracteristici aflate în alte sisteme de baze de date și pe care MySQL nu le acceptă. Secțiunea de față prezintă caracteristicile care lipsesc și indică modalități de a ocoli aceste omisiuni, acolo unde este posibil. În general, caracteristicile lipsesc din MySQL deoarece au consecințe negative asupra performanțelor. Numeroase articole din această listă se află pe lista de priorități a dezvoltatorilor, în vederea implementării în măsura timpului disponibil și cu presupunerea că pot fi implementate fără sacrificarea scopului lor inițial, acela de a obține un nivel optim de performanță.

- **Subselecție.** O subselecție este o instrucțiune `SELECT` imbricată într-o altă instrucțiune `SELECT`, cum este cazul în următoarea interogare:

  ```
  SELECT * FROM puncte
  WHERE eveniment_id IN (SELECT eveniment_id FROM eveniment WHERE tip = "T")
  ```

  Subselecțiile sunt programate să apară în MySQL versiunea 3.24, moment în care nu vor mai constitui o omisiune. Până atunci, multe interogări care sunt scrise folosind subselecții pot fi redactate alternativ sub formă de uniri. Vezi secțiunea „Rescrierea subselecțiilor sub formă de uniri".
- **Tranzacții și înregistrare-revenire.** O tranzacție este un set de instrucțiuni SQL care sunt executate unitar, fără întrerupere din partea altor clienți. Caracteristica de înregistrare-revenire vă permite să declarați că instrucțiunile trebuie executate unitar sau deloc. Cu alte cuvinte, dacă vreo instrucțiune din cadrul tranzacției eșuează, toate instrucțiunile executate până în acel moment sunt anulate.

  MySQL execută automat blocarea pentru instrucțiunile SQL individuale, pentru a preveni interacțiunile dintre clienți. (De exemplu, doi clienți nu pot scrie simultan în același tabel.) În plus, puteți folosi `LOCK TABLES` și `UNLOCK TABLES` pentru a grupa instrucțiunile într-un tot unitar, ceea ce vă permite să efectuați operații pentru care controlul concurenței pentru o singură instrucțiune nu este suficient. Problema MySQL legate de tranzacții este aceea că sistemul nu va grupa în mod automat instrucțiunile, iar dumneavoastră nu puteți efectua anularea (revenirea) instrucțiunilor dacă vreuna eșuează.

  Pentru a vedea în ce mod pot fi utile tranzacțiile, să presupunem că activați în domeniul vânzărilor de confecții și că actualizați nivelurile de inventar ori de câte ori vreunul din agenții dumneavoastră de vânzări realizează o vânzare. Exemplul următor

ilustrează tipul de probleme care pot surveni atunci când mai mulţi agenţi de vânzări actualizează simultan baza de date (presupunând că inventarul iniţial de cămăşi era de 47 de bucăţi):

t1 Agentul 1 vinde trei cămăşi

t2 Agentul 1 regăseşte numărul total de cămăşi (47):

```
SELECT cantitate FROM inventar WHERE art = "camasa"
```

t3 Agentul 2 vinde trei cămăşi

t4 Agentul 2 regăseşte numărul total de cămăşi (47):

```
SELECT cantitate FROM inventar WHERE art = "camasa"
```

t5 Agentul 1 calculează noul nivel al inventarului cu expresia 47-3=44 şi stabileşte numărul de cămăşi la 44:

```
UPDATE inventar SET cantitate = 44 WHERE art = "camasa"
```

t6 Agentul 2 calculează noul nivel al inventarului cu expresia 47-2=45 şi stabileşte numărul de cămăşi la 45:

```
UPDATE inventar SET cantitate = 45 WHERE art = "camasa"
```

La sfârşitul acestei secvenţe de evenimente, aţi vândut cinci cămăşi (asta-i bine), dar în inventar scrie 45 de bucăţi, în loc de 42 (asta nu-i bine). Problema este că, în cazul în care căutaţi valoarea inventarului într-o instrucţiune şi actualizaţi valoarea într-o altă instrucţiune, aveţi o tranzacţie cu instrucţiuni multiple. Acţiunea întreprinsă în a doua instrucţiune depinde de valoarea regăsită în prima. Dar, dacă se produc tranzacţii separate în intervale de timp care se suprapun, instrucţiunile din fiecare tranzacţie se întreţes şi interferă una cu alta. Într-o bază de date tranzacţională, instrucţiunile fiecărui agent de vânzări pot fi executate ca o tranzacţie, iar instrucţiunile Agentului 2 nu vor executa decât după finalizarea execuţiei instrucţiunilor Agentului 1. În MySQL, puteţi obţine acelaşi efect în două moduri:

- **Soluţie ocolitoare 1: Executaţi un grup de instrucţiuni ca pe un tot unitar.** Puteţi grupa instrucţiuni la un loc şi le puteţi executa ca pe o unitate indivizibilă delimitându-le cu ajutorul instrucţiunilor `LOCK TABLES` şi `UNLOCK TABLES`: blocaţi toate tabelele pe care trebuie să le folosiţi, emiteţi interogările şi anulaţi blocările. Această operaţie interzice oricui altcuiva să folosească tabelele în timp ce acestea sunt blocate. Folosind blocarea tabelelor, situaţia inventarului se prezintă astfel:

  t1 Agentul 1 vinde trei cămăşi

  t2 Agentul 1 obţine o blocare şi regăseşte numărul curent de cămăşi (47):

  ```
  LOCK TABLES inventar WRITE
  SELECT cantitate FROM inventar WHERE art = "camasa"
  ```

  t3 Agentul 2 vinde două cămăşi

  t4 Agentul 2 încearcă să obţină o blocare; această încercare va eşua, deoarece Agentul 1 deţine deja o blocare:

  ```
  LOCK TABLES inventar WRITE
  ```


  t5 Agentul 1 calculează noua valoare a inventarului cu formula 47-3=44, stabileşte numărul de cămăşi la 44 şi anulează blocarea:

  ```
  UPDATE inventar SET cantitate = 44
  WHERE art = "camasa" UNLOCK TABLES
  ```

  t6 Acum, cererea pentru o blocare, formulată de Agentul 2, reuşeşte. Agentul 2 regăseşte numărul total de cămăşi (44):

  ```
  SELECT cantitate FROM inventar WHERE art = "camasa"
  ```

  t7 Agentul 2 calculează noua valoare a inventarului cu formula 44-2=42, stabileşte numărul de cămăşi la 42 şi eliberează blocarea:

  ```
  UPDATE inventar SET cantitate = 42
  WHERE art = "camasa" UNLOCK TABLES
  ```

  Acum, instrucţiunile din cele două tranzacţii nu se amestecă una cu alta şi nivelul inventarului este stabilit corect. Vom folosi aici o blocare `WRITE`, deoarece este necesar să modificăm tabelul `inventar`. Dacă vă rezumaţi la citirea tabelelor, puteţi folosi în schimb o blocare `READ`. Aceasta permite altor clienţi să citească tabelele în timp ce dumneavoastră le folosiţi, dar îi împiedică să scrie în acestea.

  În exemplul prezentat anterior, Agentul 2 probabil că nu va sesiza nici o scădere a vitezei, deoarece tranzacţiile sunt scurte şi se vor executa rapid. Totuşi, ca regulă generală, este de dorit să evitaţi blocarea tabelelor pentru un interval lung de timp.

  Dacă folosiţi mai multe tabele, trebuie să le blocaţi pe toate înainte de a executa interogările grupate. Dacă doar citiţi dintr-un anumit tabel, totuşi, aveţi nevoie numai de o blocare a tabelului de tip citire, nu de una de tip scriere. Să presupunem că aveţi un set de interogări prin care doriţi să faceţi unele modificări într-un tabel `inventar`, şi de asemenea doriţi să citiţi unele date dintr-un tabel `client`. În acest caz, aveţi nevoie de o blocare la scriere a tabelului `inventar`, respectiv de o blocare la citire a tabelului `client`:

  ```
  LOCK TABLES inventar WRITE, client READ
  ...
  UNLOCK TABLES
  ```

  Această operaţie impune ca dumneavoastră să executaţi personal blocarea, respectiv deblocarea tabelelor. Un sistem de baze de date cu suport pentru tranzacţii va efectua aceste acţiuni în mod automat. Totuşi, aspectul privind gruparea instrucţiunilor ca un tot unitar în vederea execuţiei este acelaşi ca în cazul sistemelor de baze de date tranzacţionale.

- **Soluţie ocolitoare 2: Folosiţi actualizări relative, nu absolute.** A doua modalitate de a rezolva problema amestecului instrucţiunilor provenite din multiple tranzacţii este eliminarea dependenţei între instrucţiuni. Deşi acest lucru nu este întotdeauna posibil, vom presupune că este posibil pentru exemplul nostru cu inventarul. Pentru metoda de actualizare a inventarului folosită în soluţia ocolitoare 1, tranzacţia constă din căutarea valorii curente a inventarului, calculul noii valori în funcţie de numărul de cămăşi vândute şi apoi actualizarea cifrei de

inventar la noua valoare. Această operație este posibilă într-o singură etapă, prin simpla actualizare a numărului de cămăși relativ la valoarea sa curentă:

t1 Agentul 1 vinde trei cămăși

t2 Agentul 1 decrementează numărul de cămăși cu trei:

```
UPDATE inventar SET cantitate = cantitate - 3 WHERE art =
"camasa"
```

t3 Agentul 2 vinde două cămăși

t4 Agentul 2 decrementează numărul de cămăși cu doi:

```
UPDATE inventar SET cantitate = cantitate - 2 WHERE art =
"camasa"
```

După cum se poate vedea, această metodă nu necesită sub nici o formă tranzacții cu mai multe instrucțiuni și, ca atare, nu este necesară blocarea tabelelor pentru simularea caracteristicilor tranzacționale. Dacă tipurile de tranzacții pe care le-ați folosit până acum sunt similare celui prezentat mai sus, puteți rezolva problema fără a folosi deloc tranzacții.

Exemplul precedent indică modul de evitare a „necesității" tranzacțiilor într-o anumită situație. Aceasta nu înseamnă că nu există și alte situații când chiar *este* nevoie de tranzacții. Un exemplu caracteristic în acest sens implică un transfer financiar, când banii dintr-un cont sunt plasați într-un alt cont. Să presupunem că Marin îi scrie un cec lui Dorin pentru suma de 100 de dolari, iar Dorin încasează cecul. Contul lui Marin trebuie decrementat cu 100 de dolari, iar contul lui Dorin trebuie incrementat cu aceeași sumă:

```
UPDATE cont SET balanta = balanta - 100 WHERE nume = "Marin"
UPDATE cont SET balanta = balanta + 100 WHERE nume = "Dorin"
```

Dacă se produce o cădere a sistemului între cele două instrucțiuni, tranzacția este incompletă. Un sistem de baze de date cu caracteristici autentice de tranzacții și de înregistrare-revenire este capabil să rezolve această problemă. (Cel puțin teoretic. S-ar putea să fiți obligat să determinați tranzacțiile care nu au fost introduse și să le emiteți din nou, dar cel puțin nu mai aveți probleme cu jumătate dintre tranzacții.) În MySQL, puteți determina starea tranzacției la momentul căderii prin examinarea jurnalului de actualizare, deși acest lucru impune o oarecare examinare manuală a jurnalului.

- **Chei externe și integritate referențială.** O cheie externă vă permite să declarați că o cheie dintr-un tabel este corelată cu o cheie dintr-un alt tabel, iar integritatea referențială vă permite să impuneți restricții în ceea ce privește acțiunile care pot fi efectuate în tabelul care conține cheia externă. De exemplu, tabelul `puncte` din baza noastră de date demonstrativă `samp_db` conține o coloană `elev_id`, pe care o folosim pentru a corela înregistrările privind punctajele cu elevii din tabelul `elev`. Coloana `puncte.elev_id` ar fi declarată cheie externă, în sistemele de baze de date care acceptă acest concept, și asupra acesteia s-ar impune restricția conform căreia nu este permisă introducerea unei înregistrări cu punctajul unui elev care nu există în tabelul `elev`. În plus, se permite ștergerea în cascadă, în sensul că, dacă un elev ar fi șters din tabelul `elev`, toate înregistrările cu punctajele obținute de elevul respectiv vor fi șterse automat din tabelul `puncte`.

Cheile externe contribuie la păstrarea consecvenței datelor dumneavoastră și asigură o oarecare comoditate. Motivele pentru care aceste chei nu sunt acceptate de MySQL se datorează cu precădere unor anumite efecte negative ale cheilor externe asupra performanțelor și a întreținerii bazelor de date. (Manualul de referință MySQL conține o listă întreagă cu asemenea motive.) Rețineți că acest punct de vedere în ceea ce privește cheile externe este oarecum diferit de ceea ce veți găsi în literatura de specialitate din domeniul bazelor de date, unde cheile externe sunt descrise folosind termeni precum „esențial". Dezvoltatorii sistemului MySQL nu subscriu la acest punct de vedere. Dacă dumneavoastră o faceți, cel mai bine este să aveți în vedere alte sisteme de baze de date în vederea furnizării suportului pentru cheile externe. De exemplu, dacă între datele dumneavoastră există relații deosebit de complexe, nu doriți să fiți responsabil cu implementarea acestor dependențe în aplicațiile dumneavoastră (chiar dacă, așa cum se întâmplă frecvent, este vorba de ceva doar cu puțin mai mult decât adăugarea de instrucțiuni `DELETE` suplimentare).

MySQL nu acceptă cheile externe, în afara situației când analizează clauzele `FOREIGN KEY` în instrucțiunile `CREATE TABLE`. (Astfel, devine mai simplă portarea programelor SQL din alte sisteme de baze de date în MySQL.) MySQL nu impune cheile externe ca pe o restricție și nici nu furnizează caracteristica de ștergere în cascadă.

În mod frecvent, restricțiile pentru a căror aplicare sunt folosite cheile externe nu sunt atât de dificil de implementat prin intermediul logicii de aplicație. Uneori, nu este altceva decât o chestiune de abordare a procesului de introducere a datelor. De exemplu, pentru a introduce înregistrări noi în tabelul nostru `puncte`, este improbabilă inserția unor punctaje obținute de elevi inexistenți. Evident, modul de inserție a unui set de punctaje constă în a începe cu o listă de elevi din tabelul `elev` și apoi, pentru fiecare elev în parte, de a lua punctajul și de a folosi numărul de identificare al elevului pentru a genera o nouă înregistrare în tabelul `puncte`. Cu această procedură, nu există nici o posibilitate de a insera o înregistrare pentru un elev care nu există. Nu veți „inventa" o înregistrare cu punctaje pentru a o insera în tabelul `puncte`.

Pentru a realiza efectul ștergerilor în cascadă, trebuie să le implementați cu propria dumneavoastră logică de aplicație. Să presupunem că doriți să ștergeți elevul nr. 13. Aceasta mai înseamnă că doriți să ștergeți toate înregistrările cu punctajele obținute de elevul respectiv. Într-un sistem de baze de date care acceptă ștergerile în cascadă, ați șterge înregistrarea din tabelul `elev` și toate înregistrările din tabelul `puncte` cu următoarea instrucțiune:

```
DELETE FROM elev WHERE elev_id = 13
```

Înregistrările din tabelul `puncte` pentru elevul nr. 13 vor fi șterse automat. În MySQL, efectuați personal ștergerea secundară, cu o instrucțiune `DELETE` explicită:

```
DELETE FROM elev WHERE elev_id = 13
DELETE FROM puncte WHERE elev_id = 13
```

- **Proceduri stocate și declanșatori.** O procedură stocată este un program SQL care este compilat și stocat în server. La acest program se poate face referire ulterior, fără a mai

fi necesar ca programul să fie trimis de client şi analizat din nou. De asemenea, într-o procedură puteţi face modificări care vor afecta orice aplicaţie client care o foloseşte. Declanşatorul permite activarea unei proceduri stocate la producerea unui anumit eveniment, cum ar fi ştergerea unei înregistrări dintr-un tabel. De exemplu, puteţi recurge la această metodă dacă doriţi să regeneraţi un sumar de natură complexă din care făcea parte înregistrarea, pentru a păstra sumarul în stare actualizată. Un limbaj care să accepte proceduri stocate se află pe lista de priorităţi a dezvoltatorilor MySQL.

- **Vederi.** O *vedere* este o entitate logică, entitate care se comportă ca un tabel, dar nu este tabel. Vederea asigură o modalitate de a examina coloanele din tabele diferite ca şi cum toate acele coloane ar face parte din acelaşi tabel. Uneori, vederile se numesc tabele virtuale. Şi vederile se află pe lista de priorităţi a dezvoltatorilor sistemului MySQL.
- **Privilegii şi blocare la nivel de înregistrare.** MySQL acceptă diferite nivele de privilegii, de la privilegii globale şi până la privilegii la nivel de bază de date, tabel şi coloană. Totuşi, MySQL nu acceptă privilegii la nivel de înregistrare. Totuşi, puteţi folosi funcţiile `GET_LOCK()` şi `RELEASE_LOCK()` în aplicaţiile dumneavoastră pentru a implementa blocările cooperative la nivel de înregistrare. Procedura de aplicare a acestei metode este descrisă în rubrica aferentă funcţiei `GET_LOCK()` din Anexa C, „Referinţă de operatori şi funcţii".
- **Utilizarea caracterului - - ca şi comentariu.** Acest stil de comentariu nu este acceptat, deoarece reprezintă o construcţie ambiguă, deşi, începând cu MySQL 3.23.3, un comentariu care începe cu două liniuţe şi un spaţiu este acceptat. Vezi secţiunea „Scrierea comentariilor" pentru mai multe informaţii.

CAPITOLUL 4

Optimizarea interogărilor

Lumea teoriei bazelor de date relaţionale este o lume dominată de tabele şi seturi, precum şi de operaţii cu tabele şi seturi. O *bază de date* este un set de tabele, iar un *tabel* este un set de rânduri şi coloane. Când emiteţi o interogare `SELECT` pentru a regăsi rânduri dintr-un tabel, obţineţi un alt set de rânduri şi de coloane. Acestea sunt noţiuni abstracte, care nu fac nici un fel de referire la reprezentarea fundamentală pe care o foloseşte un sistem de baze de date pentru a opera cu datele din tabelele dumneavoastră. O altă abstractizare este aceea că operaţiile dintr-un tabel se produc toate simultan; interogările sunt conceptualizate ca fiind operaţii cu seturi, iar în teoria seturilor nu există conceptul de timp.

Lumea reală, evident, este total diferită. Sistemele de gestiune a bazelor de date implementează concepte abstracte, dar implementarea se produce folosind componente hardware reale, limitate de restricţii autentice de natură fizică. În consecinţă, interogările necesită timp – uneori un timp agasant de lung. Iar nouă, creaturi nerăbdătoare, nu ne place să aşteptăm, deci lăsăm în pace lumea abstractă a operaţiilor matematice instantanee cu seturi şi căutăm modalităţi de a mări viteza interogărilor. Din fericire, există numeroase tehnici în acest sens. Indexăm tabelele pentru a permite serverului de baze de date să caute rândurile mai rapid. Concepem interogările de aşa manieră încât să beneficiem la maximum de aceste indexuri. Scriem interogări care să influenţeze mecanismul de planificare al serverului astfel încât interogările care sosesc de la mai mulţi clienţi să coopereze mai bine. Ne gândim la ceea ce se întâmplă cu acele componente hardware de bază şi la modul în care putem ocoli restricţiile fizice impuse acestora în vederea îmbunătăţirii performanţelor.

Acestea sunt categoriile de probleme asupra cărora se va concentra capitolul de faţă, cu scopul de a vă asista la optimizarea performanţei sistemului dumneavoastră de baze de date astfel încât acesta să vă prelucreze interogările cât mai rapid posibil. MySQL este deja foarte rapid, dar chiar şi cel mai rapid sistem de baze de date poate rula interogările mai rapid dacă îl ajutaţi.

Utilizarea indexării

Vom începe cu indexarea, deoarece este cel mai important instrument pe care-l aveţi la dispoziţie pentru accelerarea interogărilor dumneavoastră. Aveţi la dispoziţie şi alte tehnici, dar, în general, elementul care va determina cea mai semnificativă diferenţă îl constituie utilizarea adecvată a indexurilor. În lista de corespondenţă MySQL, oamenii cer adesea ajutor pentru a determina o interogare să ruleze mai rapid. Într-un număr sur-

prinzător de mare de cazuri, tabelele în chestiune nu conțin nici un fel de indexuri, iar adăugarea indexurilor duce deseori la rezolvarea imediată a problemei. Această metodă nu funcționează întotdeauna, deoarece optimizarea nu este întotdeauna un proces simplu. Totuși, dacă nu folosiți indexuri, în multe situații vă pierdeți vremea încercând să îmbunătățiți performanțele prin alte mijloace. Utilizați mai întâi indexarea, pentru a obține cel mai mare salt de performanță, după care încercați să vedeți care dintre celelalte tehnici ar putea fi utilă.

Această secțiune descrie noțiunea de index, modul în care acesta îmbunătățește performanțele interogărilor, situațiile în care indexurile pot afecta în mod negativ performanțele, precum și modul de alegere a indexurilor pentru tabelele dumneavoastră. În secțiunea următoare, vom discuta despre utilitarul MySQL de optimizare a interogărilor. Pe lângă cunoașterea modului de creare a indexurilor, este bine să înțelegeți într-o oarecare măsură noțiunile privind utilitarul de optimizare, deoarece atunci veți putea beneficia mai bine de indexurile pe care le creați. Unele moduri de scriere a interogărilor anihilează utilitatea indexurilor și, în general, doriți să evitați aceste situații. (Nu întotdeauna, însă. Uneori veți dori să ignorați comportarea utilitarului de optimizare. Vom discuta și despre unele din aceste situații.)

Avantajele indexării

Să vedem cum funcționează un index pornind de la un tabel care nu are indexuri. Un tabel fără indexuri este pur și simplu o colecție dezordonată de rânduri. De exemplu, figura 4.1 prezintă tabelul `reclama` pe care l-am văzut anterior în capitolul 1, „Introducere în MySQL și SQL". Acest tabel nu conține indexuri, deci, în cazul în care căutăm rândurile aferente unei anumite companii, trebuie să examinăm fiecare rând din tabel, pentru a vedea dacă rândul respectiv conține valoarea dorită. Aceasta implică o baleiere completă a tabelului, ceea ce reprezintă o operație lentă și înfiorător de ineficientă dacă tabelul conține numai câteva înregistrări care corespund criteriilor de căutare.

Figura 4.2 prezintă același tabel, la care s-a adăugat un index pentru coloana `num_companie` din tabelul `reclama`. Indexul conține o intrare pentru fiecare rând din tabelul `reclama`, dar intrările indexului sunt sortate în funcție de valoarea din coloana `num_companie`. Acum, în loc să căutăm rând cu rând pentru a găsi articolele care corespund criteriilor de căutare, putem folosi indexul. Să presupunem că am căuta toate rândurile aferente companiei 13. Începem să căutăm în index și găsim trei rânduri aferente companiei respective. Apoi, ajungem la rândul aferent companiei 14, valoare mai mare decât cea pe care o căutăm. Valorile indexului sunt sortate, deci, atunci când citim o înregistrare care conține numărul 14, nu vom mai găsi echivalențe cu criteriile de căutare și putem sista căutarea. Dacă am fi căutat o valoare care nu apare decât undeva pe la mijlocul tabelului indexat, există algoritmi de poziționare care găsesc prima intrare de index care corespunde criteriilor de căutare fără a executa o baleiere liniară a tabelului (de exemplu o căutare binară). Astfel, ne putem poziționa rapid pe prima valoare care corespunde criteriilor de căutare și putem economisi foarte mult timp de căutare. Sistemele de baze de date folosesc diverse tehnici de poziționare pentru indexarea rapidă a valorilor, dar deocamdată nu ne interesează tehnicile respective. Este important că ele funcționează și că indexarea este un lucru bun.

Poate vă întrebați: de ce nu se sortează fișierul de date, după care se șterge fișierul de index? Nu s-ar produce aceeași îmbunătățire a vitezei de căutare? Aveți dreptate, s-ar produce – dacă ați fi avut un singur index. Dar puteți dori un al doilea index și nu puteți sorta fișierul de date în două moduri diferite simultan. (De exemplu, doriți un index după numele clienților și un alt index după numerele de identificare sau numerele de telefon ale acestora.) Utilizarea indexurilor ca entități separate de fișierul de date rezolvă problema și permite crearea mai multor indexuri. De asemenea, rândurile din index sunt, în general, mai scurte decât rândurile de date. Când inserați sau ștergeți valori noi, este mai ușor să mutați valori scurte din index pentru a păstra ordinea de sortare decât să vă deplasați în rândurile de date mai lungi.

tabelul reclama

num_companie	num_reclama	taxa_deschidere
14	48	0.01
23	49	0.02
17	52	0.01
13	55	0.03
23	62	0.02
23	63	0.01
23	64	0.02
13	77	0.03
23	99	0.03
14	101	0.01
13	102	0.01
17	119	0.02

Figura 4.1 Tabelul `reclama` fără index.

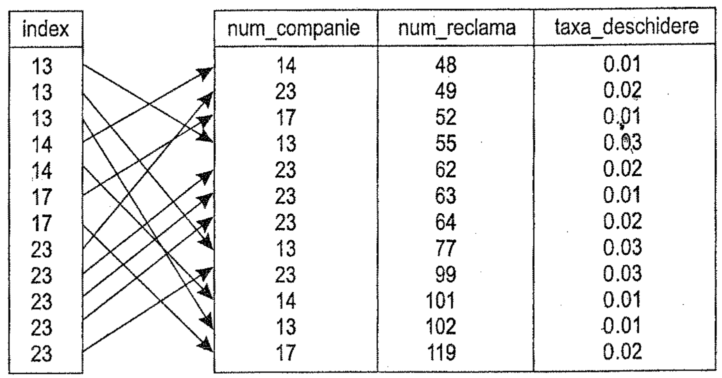


Figura 4.2 Tabelul `reclama` indexat.

Exemplul corespunde modului de indexare a tabelelor în MySQL. Rândurile de date ale unui tabel sunt păstrate într-un fișier cu date, iar valorile din index sunt păstrate într-un fișier index. Puteți avea mai multe indexuri într-un tabel; în acest caz, toate indexurile sunt stocate în același fișier index. Fiecare index din fișierul index constă dintr-un tablou sortat cu înregistrări cheie care sunt folosite pentru un acces rapid la fișierul de date.

Expunerea anterioară a prezentat avantajele unui index în contextul interogărilor într-un singur tabel, acolo unde utilizarea unui index mărește semnificativ viteza de căutare, prin eliminarea necesității de baleiere completă a tabelului. Totuși, indexurile sunt și mai valoroase atunci când rulați interogări care implică uniri pe tabele multiple. Într-o interogare cu un singur tabel, numărul de valori pe care trebuie să le examinați în fiecare coloană este egal cu numărul de înregistrări din tabel. Într-o interogare pe tabele multiple, numărul combinațiilor posibile crește amețitor, deoarece este egal cu produsul numerelor de rânduri ale tabelelor.

Să presupunem că aveți trei tabele fără index, și anume `t1`, `t2` și `t3`, conținând respectiv coloanele `c1`, `c2` și `c3` și câte 1000 de rânduri care conțin numerele cuprinse între 1 și 1000. (Exemplul este exagerat, desigur. Nu-l folosim decât pentru a trage o concluzie; cu toate acestea, problemele pe care le vom ilustra sunt reale.) O interogare pentru regăsirea tuturor combinațiilor între acele rânduri ale tabelelor care conțin valori egale arată astfel:

```
SELECT c1, c2, c3
FROM t1, t2, t3
WHERE c1 = c2 AND c1 = c3
```

Rezultatul acestei interogări trebuie să fie compus din 1000 de rânduri, fiecare rând conținând trei valori egale. Dacă prelucrăm interogarea în absența indexurilor, nu vom avea nici o idee cu privire la valorile incluse în fiecare rând. În consecință, trebuie să încercăm toate combinațiile pentru a le găsi pe cele care corespund criteriilor din clauza `WHERE`. Numărul de combinații posibile este 1000 × 1000 × 1000 (1 miliard!), adică de un milion de ori mai mare decât numărul rândurilor care corespund criteriilor de căutare. Asta înseamnă un volum extrem de mare de efort irosit și probabil că această interogare va fi foarte lentă, chiar și pentru un sistem de baze de date precum MySQL, care este foarte rapid. Iar asta se întâmplă pentru un tabel cu numai 1000 de rânduri. Ce ne facem în situațiile când aveți tabele cu milioane de rânduri? Puteți vedea că astfel se ajunge foarte rapid la performanțe extrem de reduse. Dacă indexăm fiecare tabel, putem mări considerabil viteza de prelucrare a interogărilor, deoarece indexarea permite prelucrarea interogării în acest mod:

1. Selectează primul rând din tabelul `t1` și citește valoarea pe care o conține rândul.
2. Folosind indexul din tabelul `t2`, se deplasează direct la rândul care corespunde valorii din tabelul `t1`. Similar, folosind indexul din tabelul `t3`, se deplasează direct la rândul care corespunde valorii din tabelul `t1`.
3. Trece la următorul rând din tabelul `t1` și repetă procedura anterioară până când examinează toate rândurile din tabelul `t1`.

În acest caz, se execută totuși o parcurgere completă a tabelului `t1`, dar putem executa căutări indexate în tabelele `t2` și `t3` pentru a extrage direct rândurile din aceste tabele. În acest mod, interogarea rulează efectiv de un milion de ori mai repede.

MySQL folosește indexurile așa cum s-a arătat mai sus, pentru a mări viteza de căutare a rândurilor care satisfac criteriile unei clauze `WHERE`, respectiv a rândurilor care corespund rândurilor din alte tabele, atunci când se efectuează uniri. De asemenea, MySQL folosește indexuri și pentru a îmbunătăți performanțele altor tipuri de operații:

- Valoarea cea mai mică sau cea mai mare a unei coloane indexate se poate găsi rapid, fără a se examina fiecare rând atunci când folosiți funcțiile `MIN()` sau `MAX()`.
- MySQL poate folosi frecvent indexuri pentru a efectua rapid operații de sortare pentru clauzele `ORDER BY`.
- Uneori, MySQL poate evita complet citirea fișierului de date. Să presupunem că selectați valori dintr-o coloană numerică indexată și că nu selectați alte coloane din tabel. În acest caz, prin citirea unei valori din index, ați obținut deja valoarea care ar fi rezultat din citirea fișierului de date. Nu are nici un sens să citiți valorile de două ori, deci nu este necesară nici măcar consultarea fișierului de date.

Dezavantajele indexării

În general, dacă MySQL poate determina modul de utilizare a unui index pentru a prelucra o interogare mai rapid, o va face. Aceasta înseamnă că, în majoritatea cazurilor, dacă nu vă indexați tabelele, ieșiți în pierdere. Ați văzut că am zugrăvit o imagine roză a avantajelor indexării. Există și dezavantaje? Da, există. În practică, aceste neajunsuri tind să fie contrabalansate de avantaje, dar trebuie să le cunoașteți și pe acestea.

Mai întâi, fișierul index ocupă spațiu pe disc. Dacă aveți numeroase indexuri, fișierul index poate ajunge la dimensiunea maximă mai rapid decât fișierul de date. În al doilea rând, indexurile măresc viteza operațiilor de regăsire, dar o reduc pe aceea a operațiilor de inserție și ștergere, precum și viteza operațiilor de actualizare a valorilor din coloanele indexate (adică majoritatea operațiilor care implică scriere), deoarece o scriere afectează nu numai rândul de date, dar și indexurile, în majoritatea cazurilor. Cu cât un tabel are un număr mai mare de indexuri, cu atât nivelul mediu de degradare a performanțelor pentru operațiile de scriere este mai mare. În secțiunea „Încărcarea eficientă a datelor" vom aborda mai detaliat această problemă de performanță și modalitățile de rezolvare.

Alegerea indexurilor

Sintaxa de creare a indexurilor a fost prezentată în secțiunea „Crearea și ștergerea indexurilor" din capitolul 3, „Sintaxa și utilizarea SQL în MySQL". Voi presupune că ați citit secțiunea respectivă. Dar simpla cunoaștere a sintaxei nu vă ajută să determinați modul în care trebuie indexate tabelele dumneavoastră. Acesta impune o gândire a modului în care vă veți folosi tabelele. Această secțiune oferă unele indicații în ceea ce privește identificarea și selectarea coloanelor adecvate pentru indexare:

- **Indexați coloanele pe care le căutați, nu coloanele pe care le selectați.** Cu alte cuvinte, coloanele cele mai potrivite pentru indexare sunt coloanele care apar în clauza `WHERE`, respectiv coloanele specificate în clauzele de unire, nu coloanele care apar în lista de selecție imediat următoare cuvântului cheie `SELECT`:

```
SELECT
    col_a                                ⇦ coloana nu este adecvată
FROM
    tabel1 LEFT JOIN tabel2
    ON tabel1.col_b = tabel2.col_c       ⇦ coloane adecvate
```

```
WHERE
    col_d = expresie                ⇦ o coloană adecvată
```

Coloanele pe care le selectați și coloanele pe care le folosiți în clauza WHERE pot fi identice, desigur. Ideea este că apariția unei coloane în lista de selecție nu reprezintă o sugestie pentru indexarea acesteia.

Coloanele care apar în clauzele de unire sau în expresii de forma `col1 = col2` sunt extrem de adecvate pentru indexare. Coloanele `col_b` și `col_c` din interogarea prezentată anterior sunt exemple în acest sens. Dacă MySQL poate optimiza o interogare folosind coloane unite, atunci reduce semnificativ posibilele combinații tabel-rând prin eliminarea baleierilor complete ale tabelului.

- **Folosiți indexuri unice.** Gândiți-vă la domeniul de valori al unei coloane. Indexurile funcționează cel mai bine pentru coloane cu valori unice, respectiv cel mai slab în cazul coloanelor care prezintă numeroase valori duplicate. De exemplu, dacă o coloană conține vârste și are numeroase valori diferite, un index va face imediat diferența între rânduri. Un index nu va fi de mare ajutor dacă o coloană este folosită pentru a înregistra sexul unei persoane și conține numai cele două valori "M" și "F" (indiferent de valoarea pe care o căutați, oricum veți obține cam jumătate din rânduri).
- **Folosiți indexuri scurte.** Dacă indexați o coloană șir, specificați o lungime a prefixului ori de câte ori situația o permite. De exemplu, dacă aveți o coloană `CHAR(200)`, nu indexați întreaga coloană dacă primele 10 sau 20 de caractere dintr-o valoare sunt unice. Prin indexarea primelor 10 sau 20 de caractere se va economisi o mare cantitate de spațiu în fișierul index și probabil că interogările vor deveni și ele mai rapide. Un index mai mic necesită mai puține operații de intrare-ieșire cu discul, iar valorile mai scurte pot fi comparate mai rapid. Mai important este că, dacă valorile cheilor sunt mai scurte, blocurile din zona cache a indexului pot conține mai multe valori ale cheilor, deci MySQL poate stoca în memorie mai multe chei simultan. Astfel, crește probabilitatea de localizare a rândurilor fără a mai fi necesară citirea de blocuri suplimentare din index. (Doriți să vă folosiți și bunul simț, desigur. Indexarea numai a primului caracter dintr-o coloană nu este chiar atât de utilă, deoarece în index nu vor exista prea multe valori distincte.)
- **Utilizați prefixele de stânga.** Când creați un index cu n coloane, de fapt creați n indexuri pe care MySQL le poate folosi. Un index pentru mai multe coloane este echivalent cu mai multe indexuri, deoarece orice set de coloane de stânga ale indexului se poate folosi pentru stabilirea corespondenței cu anumite rânduri. Un asemenea set se numește prefix de stânga[1] (în original *leftmost prefix* – N.T.).

  Să presupunem că aveți un tabel cu un index pentru coloane denumite `stat`, `oras` și `cod_postal`. Rândurile din index sunt sortate în ordinea `stat/oras/cod_postal`, deci sunt automat sortate atât în ordinea `stat/oras`, cât și în funcție de stat. Aceasta înseamnă că MySQL poate beneficia de index chiar dacă specificați într-o interogare numai valorile din coloana `stat`, respectiv numai valorile din coloanele `stat` și `oras`. Astfel, indexul poate fi folosit pentru a căuta următoarele combinații de coloane:

[1] Această operație este diferită de indexarea prefixului unei coloane, care folosește primele n caractere ale coloanei pentru valorile din index. – N.A.


```
stat, oras, cod_postal
stat, oras
stat
```

  MySQL nu poate folosi indexul pentru căutări care nu implică un prefix de stânga. De exemplu, în cazul în care căutați în funcție de `oras` sau `cod_postal`, indexul nu va fi folosit. Dacă sunteți în căutarea unui stat dat și al unui anumit cod poștal (coloanele 1 și 3 ale indexului), indexul nu poate fi folosit pentru combinația de valori respectivă. Totuși, MySQL poate reduce domeniul de căutare folosind indexul pentru a găsi rândurile care conțin statul căutat.
- **Nu folosiți indexul în mod excesiv.** Nu indexați tot ce vedeți cu ochii, pornind de la zicala „cu cât mai mult, cu atât mai bine". Este o greșeală. Fiecare index suplimentar ocupă spațiu pe disc și afectează performanțele operațiilor de scriere, așa cum s-a arătat deja. Indexurile trebuie actualizate și eventual reorganizate atunci când modificați conținutul tabelelor dumneavoastră, operație care are o durată direct proporțională cu numărul indexurilor. Dacă aveți un index care este folosit rareori sau niciodată, diminuați în mod inutil viteza de efectuare a modificărilor din tabel. În plus, MySQL ia în considerare indexurile atunci când generează un plan de execuție pentru regăsiri. Crearea de indexuri suplimentare înseamnă mai multă muncă pentru utilitarul de optimizare a interogărilor. Este de asemenea posibil (deși improbabil) ca MySQL să nu aleagă cel mai bun index pentru utilizare atunci când aveți prea multe indexuri. Păstrarea numai a indexurilor de care aveți nevoie ajută utilitarul de optimizare a interogărilor să evite asemenea greșeli.

  Dacă vă gândiți să adăugați un index la un tabel care este deja indexat, gândiți-vă dacă indexul pe care vă gândiți să-l adăugați nu este un prefix de stânga al unui index pe mai multe coloane deja existent. În acest caz, nu vă mai deranjați să adăugați indexul, deoarece, în realitate, îl aveți.
- **Țineți cont de tipurile de comparații pe care le efectuați cu valorile dintr-o coloană.** Indexurile sunt folosite pentru operațiile <, <=, =, >=, > și `BETWEEN`. Indexurile sunt de asemenea folosite pentru operațiile `LIKE`, atunci când modelul are un prefix literal. Dacă folosiți o coloană numai pentru alte categorii de operații, cum ar fi `STRCMP()`, nu are nici un sens să o indexați.

Utilitarul de optimizare a interogărilor din MySQL

Când emiteți o interogare pentru selectarea rândurilor, MySQL o analizează pentru a vedea dacă poate beneficia de optimizări care să-i permită să prelucreze interogarea mai rapid. În această secțiune, vom examina modul de funcționare al utilitarului de optimizare. Pentru informații suplimentare, consultați capitolul „Obținerea unui nivel maxim de performanță din utilizarea sistemului MySQL" din manualul de referință MySQL. Capitolul respectiv descrie diferite măsuri de optimizare pe care le ia MySQL. La acest capitol sunt adăugate ocazional alte informații, deoarece dezvoltatorii MySQL continuă să îmbunătățească utilitarul de optimizare, deci merită să recitiți capitolul din când în când, pentru a vedea dacă n-au apărut ponturi noi pe care să le utilizați. (Versiunea electronică a manualului MySQL, de la adresa `http://www.mysql.com/`, este în permanență actualizată.)

Utilitarul de optimizare a interogărilor din MySQL utilizează indexurile, desigur, dar folosește și alte informații. De exemplu, dacă emiteți următoarea interogare, MySQL o va executa foarte rapid, indiferent de dimensiunea tabelului:

```
SELECT * FROM nume_tabel WHERE 1 = 0
```

În acest caz, MySQL examinează clauza WHERE, sesizează că nici un rând nu poate satisface interogarea și nici măcar nu se mai obosește să caute în tabel. Puteți vedea acest lucru cu ajutorul instrucțiunii EXPLAIN, care cere sistemului MySQL să afișeze unele informații despre modul în care ar executa o instrucțiune SELECT, fără a o executa efectiv. Pentru a folosi EXPLAIN, pur și simplu inserați cuvântul EXPLAIN în fața instrucțiunii SELECT:

```
EXPLAIN SELECT * FROM nume_tabel WHERE 1 = 0
```

Rezultatul instrucțiunii EXPLAIN este următorul:

Comment
Impossible WHERE (clauză WHERE imposibila)

În mod normal, EXPLAIN returnează mai multe informații decât cea prezentată mai sus, inclusiv date despre indexurile care vor fi folosite pentru baleierea tabelelor, tipurile de uniri care vor fi folosite, respectiv estimează numărul de rânduri care vor trebui parcurse din fiecare tabel.

Modul de funcționare a utilitarului de optimizare

Utilitarul de optimizare a interogărilor din MySQL are numeroase scopuri, dar principala sa menire este de a folosi indexurile ori de câte ori este posibil, precum și de a folosi indexul cel mai restrictiv, pentru a elimina cât mai multe rânduri într-un timp cât mai scurt. Poate părea ciudat, deoarece, atunci când emiteți instrucțiuni SELECT, scopul *dumneavoastră* este de a găsi rânduri, nu de a le elimina. Motivul pentru care utilitarul de optimizare funcționează în acest mod este următorul: cu cât pot fi scoase din discuție rânduri într-o manieră mai expeditivă, cu atât pot fi găsite mai rapid rândurile care satisfac criteriile de căutare. Interogările pot fi prelucrate mai rapid dacă testele cele mai restrictive se pot efectua primele. Să presupunem că aveți o interogare care testează două coloane, ambele coloane fiind dotate cu index:

```
WHERE col1 = "o valoare" AND col2 = "o alta valoare"
```

Să presupunem că testul aplicat asupra coloanei 1 găsește 900 de rânduri, că testul aplicat asupra coloanei 2 găsește 300 de valori, și că ambele teste reușesc în cazul a 30 de rânduri. Dacă testați mai întâi coloana 1, trebuie să examinați 900 de rânduri pentru a le găsi pe cele 30 care corespund și valorii din coloana 2. Asta înseamnă 870 de teste ratate. Dacă testați mai întâi coloana 2, trebuie să examinați numai 300 de rânduri pentru a le găsi pe cele 30 care de asemenea corespund valorii din coloana 1. Numărul testelor ratate în acest caz este de 270, ceea ce implică mai puține calcule și operații de intrare-ieșire cu discul.

Puteți ajuta utilitarul de optimizare să folosească indexurile folosind următoarele îndrumări:

- **Comparați coloane de același tip.** Când folosiți coloane indexate în comparații, utilizați coloane de același tip. De exemplu, CHAR(10) este considerat identic cu CHAR(10) sau VARCHAR(10), dar diferit de CHAR(12) sau VARCHAR(12). INT este diferit de BIGINT.

Utilizarea coloanelor de același tip este o cerință obligatorie în versiunile de MySQL anterioare versiunii 3.23, în caz contrar indexurile pentru coloane nefiind folosite. Începând de la versiunea 3.23, acest lucru nu este strict necesar, dar tipurile de coloane identice vă vor oferi performanțe superioare tipurilor de coloane diferite. În cazul în care coloanele pe care le comparați sunt de tipuri diferite, puteți folosi ALTER TABLE în scopul de a modifica unul dintre ele, pentru a obține asemănarea.

- **Încercați să singularizați coloanele indexate în comparații.** Dacă folosiți o coloană într-un apel la o funcție sau într-o expresie aritmetică, MySQL nu poate folosi indexul, deoarece trebuie să calculeze valoarea expresiei pentru fiecare rând. Uneori acest lucru este inevitabil, dar de multe ori puteți rescrie o interogare pentru a obține chiar coloana indexată.

Următoarele clauze WHERE ilustrează acest aspect. În prima linie, utilitarul de optimizare va simplifica expresia 4/2 la valoarea 2, apoi va folosi un index pe coloana_mea pentru a descoperi rapid valorile mai mici decât 2. În a doua expresie, MySQL trebuie să regăsească valoarea din coloana_mea pentru fiecare rând, să multiplice cu 2 și apoi să compare rezultatul cu 4. Nu se poate folosi nici un index, deoarece trebuie regăsită fiecare valoare din coloană, astfel încât expresia din membrul stâng al comparației să poată fi evaluată:

```
WHERE coloana_mea < 4 / 2
WHERE coloana_mea * 2 < 4
```

Să ne gândim la un alt exemplu. Să presupunem că aveți o coloană indexată data_col. Dacă emiteți o interogare precum următoarea, indexul nu este folosit:

```
SELECT * FROM tabelul_meu WHERE YEAR(data_col) < 1990
```

Expresia nu compară valoarea dintr-o coloană indexată cu 1990; se compară o valoare calculată din valoarea coloanei, iar valoarea respectivă trebuie calculată pentru fiecare rând. Ca rezultat, indexul pe coloana data_col nu poate fi folosit. Care este remediul? Folosiți o dată literală, și indexul pentru coloana data_col va fi utilizat:

```
WHERE data_col < "1990-01-01"
```

Dar să presupunem că nu aveți o dată precizată. În schimb, sunteți interesat în găsirea înregistrărilor a căror dată se află la un anumit număr de zile de data de astăzi. Există numeroase metode de scriere a unei interogări precum aceasta, dar nu toate la fel de bune. Sunt trei posibilități:

```
WHERE TO_DAYS(data_col) - TO_DAYS(CURRENT_DATE) < limita
WHERE TO_DAYS(data_col) < limita + TO_DAYS(CURRENT_DATE)
WHERE data_col < DATE_ADD(CURRENT_DATE, INTERVAL limita DAY)
```

Pentru prima linie, nu va fi folosit nici un index, deoarece coloana trebuie regăsită pentru fiecare rând, pentru a permite calculul valorii TO_DAYS(data_col). A doua linie este mai bună. Atât limita cât și TO_DAYS(CURRENT_DATE) sunt constante, deci membrul drept al comparației poate fi calculat o singură dată de utilitarul de optimizare înainte de prelucrarea interogării, nu pentru fiecare rând în parte. Dar coloana data_col apare într-un apel la o funcție, deci indexul nu este folosit. Cea de-a treia linie este metoda ideală. Din nou, membrul drept al comparației poate fi calculat o singură dată, fiind o constantă, înaintea execuției interogării, dar acum valoarea este o

dată calendaristică. Acea valoare poate fi comparată direct cu valorile din coloana `data_col`, care nu mai trebuie convertite în zile. În acest caz, se poate folosi indexul.

- **Nu folosiți caractere de înlocuire la începutul unui model** `LIKE`. Uneori oamenii caută șiruri folosind o clauză `WHERE` de forma următoare:

```
WHERE nume_coloana LIKE "%sir%"
```

Aceasta este o operație corectă dacă doriți să găsiți șirul `sir` indiferent unde apare acesta în interiorul coloanei. Dar nu inserați caracterul `%` de ambele părți doar din obișnuință. Dacă într-adevăr căutați șirul numai atunci când acesta apare la începutul coloanei, atunci nu mai scrieți primul caracter `%`. De exemplu, în cazul în care căutați într-o coloană care conține nume de familie acele nume care încep cu `"Mac"`, scrieți clauza `WHERE` astfel:

```
WHERE nume LIKE "Mac%"
```

Utilitarul de optimizare examinează partea literală inițială a modelului și folosește indexul pentru a găsi rândurile care corespund acestuia, ca și cum ați fi scris următoarea expresie, care se află într-o formă ce permite utilizarea unui index pentru coloana `nume`:

```
WHERE nume >= "Mac" AND nume < "Mad"
```

Această optimizare nu se aplică operațiilor de stabilire a corespondențelor cu un model care folosesc operatorul `REGEXP`.

- **Ajutați utilitarul de optimizare să efectueze estimări mai bune ale eficienței indexului.** În mod prestabilit, când comparați valorile din coloanele indexate cu o constantă, utilitarul de optimizare va presupune că valorile cheilor sunt distribuite în mod uniform în index. De asemenea, utilitarul de optimizare va verifica rapid indexul, pentru a estima numărul de intrări din index care vor fi folosite atunci când se determină dacă indexul trebuie folosit sau nu pentru comparațiile cu valori constante. Puteți furniza utilitarului de optimizare informații mai bune folosind opțiunea `--analyze` cu `myisamchk` sau `isamchk`, pentru a analiza distribuția valorilor cheilor. Folosiți `myisamchk` pentru tabele MyISAM, respectiv `isamchk` pentru tabele ISAM. Pentru a efectua analiza cheilor, trebuie să puteți deschide o sesiune de lucru la gazda serverului MySQL și trebuie să aveți acces de scriere la fișierele tabelului.

- **Utilizați instrucțiunea** `EXPLAIN` **pentru a verifica funcționarea utilitarului de optimizare.** Verificați dacă în interogările dumneavoastră sunt folosite indexurile pentru eliminarea cu rapiditate a rândurilor. Dacă nu, puteți încerca să folosiți `STRAIGHT_JOIN` pentru a forța efectuarea unei uniri folosind tabele într-o anumită ordine. Rulați interogarea în diferite moduri pentru a vă forma o opinie; s-ar putea ca MySQL să aibă un motiv solid pentru a nu folosi indexurile în ordinea pe care dumneavoastră o considerați cea mai bună.

- **Testați forme alternative ale interogărilor, dar rulați-le de mai multe ori.** Când testați formele alternative ale unei interogări, rulați-le de mai multe ori în fiecare formă. Dacă rulați ambele variante ale unei interogări câte o singură dată pe fiecare, veți descoperi frecvent că a doua variantă este mai rapidă, și aceasta numai deoarece informațiile de la prima interogare se află încă în zona cache a discului și nu trebuie să fie efectiv citite de pe disc. De asemenea, trebuie să încercați să rulați interogările atunci când încărcarea sistemului este relativ stabilă, pentru a evita efectele datorate altor activități din sistemul dumneavoastră.

Anularea efectelor optimizării

Poate părea ciudat, dar există situații când doriți să contracarați tehnicile de optimizare ale sistemului MySQL. Unele din aceste circumstanțe sunt descrise în secțiunea de față:

- **Pentru a forța sistemul MySQL să șteargă încet conținutul unui tabel.** Când doriți să ștergeți complet conținutul unui tabel, acesta poate dispărea cel mai repede dacă se folosește o instrucțiune `DELETE` fără nici o clauză `WHERE`:

```
DELETE FROM nume_tabel
```

MySQL optimizează acest caz special de instrucțiune `DELETE`; pur și simplu re-creează fișiere de date și fișiere index vide pornind de la zero, folosind descrierea tabelului din fișierul cu informații privind tabelul. Această optimizare mărește extrem de mult viteza operației de ștergere, deoarece MySQL nu mai trebuie să șteargă fiecare rând în parte. Totuși, operația are unele efecte care nu sunt de dorit în anumite circumstanțe:

  - MySQL raportează numărul rândurilor afectate de instrucțiune ca fiind zero, chiar dacă tabelul nu era vid. În majoritatea cazurilor, acest aspect nu contează (deși poate fi derutant dacă nu vă așteptați la așa ceva), dar pentru aplicații care au efectiv nevoie să cunoască numărul real de rânduri, această opțiune nu este adecvată.
  - Dacă tabelul conține o coloană `AUTO_INCREMENT`, numerotarea secvenței din coloană este reinițializată pentru a începe de la 1. Acest lucru este adevărat chiar și în condițiile îmbunătățirii metodelor de manipulare a atributului `AUTO_INCREMENT`, introduse în MySQL versiunea 3.23 și expuse în secțiunea „Lucrul cu secvențe" a capitolului 2, „Lucrul cu date în MySQL".

Puteți „deoptimiza" o instrucțiune `DELETE` adăugând o clauză `WHERE 1>0`:

```
DELETE FROM nume_tabel WHERE 1 > 0
```

Această instrucțiune forțează sistemul MySQL să execute o ștergere rând cu rând. Interogarea se execută mult mai lent, dar va returna numărul real de rânduri șterse. De asemenea, va conserva numerotarea curentă a secvenței `AUTO_INCREMENT`, deși numai pentru tabelele MyISAM (disponibile în MySQL versiunea 3.23 și în versiunile ulterioare). Pentru tabelele ISAM, din păcate, secvența este totuși reinițializată.

- **Pentru a evita un ciclu de actualizare infinit.** Dacă actualizați o coloană indexată, este posibil ca rândurile care sunt actualizate să fie actualizate la infinit, în cazul în care coloana este folosită în clauza `WHERE` și când actualizarea mută valoarea indexului în partea din domeniu care nu a fost încă prelucrată. Să presupunem că tabelul `tabelul_meu` are o coloană întreagă `col_cheie` care este indexată. Interogări ca următoarea pot cauza probleme:

```
UPDATE tabelul_meu SET col_cheie = col_cheie+1
WHERE col_cheie > 0;
```

Soluția în acest caz este utilizarea coloanei `col_cheie` într-o expresie din clauza `WHERE`, astfel încât MySQL să nu poată folosi indexul:

```
UPDATE tabelul_meu SET col_cheie = col_cheie+1
WHERE col_cheie+0 > 0;
```

De fapt, mai există o soluție: treceți la versiunea MySQL 3.23.2 sau la o versiune mai recentă, ceea ce va avea ca efect remedierea problemei.

- **Pentru a regăsi rezultate într-o ordine aleatoare.** Începând de la versiunea MySQL 3.23.2, puteți folosi funcția `ORDER BY RAND()` pentru sortarea aleatoare a rezultatelor. O altă tehnică, utilă pentru versiuni mai vechi de MySQL, este de a selecta o coloană de numere aleatoare și de a sorta în funcție de coloana respectivă. Totuși, dacă scrieți interogarea după cum urmează, utilitarul de optimizare contracarează intenția dumneavoastră:

  ```
  SELECT ..., RAND() AS rand_col FROM ... ORDER BY rand_col
  ```

  Problema în acest caz este că MySQL „vede" că respectiva coloană este un apel la o funcție, crede că valoarea coloanei va fi o constantă și optimizează clauza `ORDER BY` chiar din interogare! Puteți induce în eroare utilitarul de optimizare făcând referire în interiorul expresiei la o coloană din tabel. De exemplu, dacă tabelul dumneavoastră are o coloană denumită `varsta`, puteți scrie interogarea astfel:

  ```
  SELECT varsta*0+RAND() AS rand_col
  FROM ... ORDER BY rand_col
  ```

- **Pentru a anula ordinea de unire a tabelelor stabilită de utilitarul de optimizare.** Folosiți `STRAIGHT_JOIN` pentru a forța utilitarul de optimizare să folosească tabelele într-o anumită ordine. Dacă procedați astfel, trebuie să ordonați tabelele de așa manieră încât primul tabel să fie cel din care va fi selectat cel mai mic număr de rânduri. (Dacă nu sunteți sigur care este acest tabel, treceți tabelul cu numărul cel mai mare de rânduri pe prima poziție din listă.) Cu alte cuvinte, încercați să ordonați tabelele astfel încât selecția cea mai restrictivă să se aplice prima. Interogările se desfășoară mai bine dacă puteți elimina cât mai rapid rândurile candidate posibile. Nu uitați să încercați interogarea în ambele sensuri; pot exista motive pentru care utilitarul de optimizare nu unește tabelele în modul în care credeți că ar trebui să procedeze, iar instrucțiunea `STRAIGHT_JOIN` poate să nu fie de prea mare ajutor.

Alegerea tipurilor de coloană și eficiența interogărilor

Această secțiune furnizează unele instrucțiuni pentru alegerea coloanelor care pot contribui la executarea mai rapidă a interogărilor[2]:

- **Folosiți coloane cu lungime fixă, nu coloane cu lungime variabilă.** Acest lucru este valabil mai ales pentru tabelele care sunt modificate frecvent și care sunt, ca atare, mai supuse la fragmentare. De exemplu, transformați toate coloanele caracter în coloane `CHAR`, nu `VARCHAR`. Compromisul este acela că tabelul dumneavoastră va folosi mai mult spațiu, dar, dacă vă puteți permite spațiul suplimentar, rândurile cu lungime fixă pot fi prelucrate mai rapid decât rândurile cu lungime variabilă.

[2] În această expunere, prin „tipuri BLOB" se vor înțelege atât tipurile BLOB, cât și tipurile TEXT. – N.A.

- **Nu folosiți coloane mai lungi atunci când se pot utiliza și coloane mai scurte.** Dacă folosiți coloane `CHAR` de lungime fixă, nu le faceți inutil de lungi. Dacă valoarea cea mai scurtă pe care o stocați într-o coloană este de 40 caractere, nu o declarați sub forma `CHAR(255)`; declarați-o sub forma `CHAR(40)`. Dacă puteți folosi `MEDIUMINT` în loc de `BIGINT`, tabelul dumneavoastră va fi mai mic (ceea ce implică mai puține operații de intrare-ieșire cu discul), iar valorile vor fi prelucrate mai rapid în cadrul calculelor.
- **Declarați coloanele ca `NOT NULL`.** Astfel, obțineți o viteză de prelucrare mai mare și aveți nevoie de un spațiu de stocare mai redus. De asemenea, va simplifica uneori interogările, deoarece nu trebuie să tratați `NULL` ca pe un caz special.
- **Luați în calcul utilizarea de coloane `ENUM`.** Dacă aveți o coloană șir care conține numai un număr limitat de valori distincte, luați în considerare conversia acesteia la o coloană `ENUM`. Valorile `ENUM` pot fi prelucrate rapid, deoarece sunt reprezentate intern sub formă de valori numerice.
- **Folosiți `PROCEDURE ANALYSE()`.** Dacă dispuneți de MySQL versiunea 3.23 sau de o versiune ulterioară, rulați funcția `PROCEDURE ANALYSE()` pentru a primi informații despre coloanele din tabelul dumneavoastră:

  ```
  SELECT * FROM nume_tabel PROCEDURE_ANALYSE()
  SELECT * FROM nume_tabel PROCEDURE_ANALYSE(16,256)
  ```

  Una dintre coloanele datelor de ieșire este o sugestie a tipului optim de coloană pentru fiecare dintre coloanele tabelului dumneavoastră. Cel de-al doilea exemplu indică funcției `PROCEDURE ANALYSE()` să nu sugereze tipuri `ENUM` care conțin mai mult de 16 valori sau care ocupă mai mult de 256 octeți (puteți modifica valorile după cum doriți). Fără asemenea restricții, datele de ieșire pot fi foarte lungi; declarațiile `ENUM` sunt deseori dificil de citit.

  În funcție de datele de ieșire ale funcției `PROCEDURE ANALYSE()`, puteți descoperi că tabelul dumneavoastră poate fi modificat pentru a beneficia de un tip mai eficient. Folosiți `ALTER TABLE` dacă doriți să modificați un tip de coloană.
- **„Împachetați" date într-o coloană `BLOB`.** Utilizarea unui tip `BLOB` pentru a stoca date pe care le împachetați și le despachetați în aplicația dumneavoastră vă poate permite să obțineți toate datele printr-o singură operație de regăsire, în loc de mai multe asemenea operații. Utilizarea tipului `BLOB` poate fi de asemenea de folos pentru datele care nu sunt ușor de reprezentat într-o structură de tabel standard sau care se modifică în timp. În expunerea instrucțiunii `ALTER TABLE` din capitolul 3, unul din exemple aborda un tabel folosit pentru stocarea rezultatelor din câmpurile unui chestionar bazat pe Web. Exemplul respectiv prezenta modul în care puteți folosi instrucțiunea `ALTER TABLE` pentru a adăuga coloane la tabel ori de câte ori adăugați întrebări la chestionar.

  O altă modalitate de abordare a acestei probleme este de a determina programul de aplicație care prelucrează formularul Web să „împacheteze" datele într-un fel de structură de date, iar apoi să le insereze într-o singură coloană `BLOB`. Astfel, aplicația capătă suprasarcina de codificare a datelor (și de decodificare a acestora mai târziu, atunci când regăsiți înregistrări din tabel), dar se simplifică structura tabelului și se elimină necesitatea de modificare a structurii tabelului atunci când chestionarul se modifică.

Pe de altă parte, valorile BLOB pot provoca propriile lor probleme, mai ales dacă efectuați o mulțime de operații de tip DELETE și UPDATE. Ștergerea unui BLOB poate lăsa un mare gol în tabel, gol care va fi umplut ulterior cu o înregistrare sau înregistrări de dimensiuni probabil diferite.

- **Folosiți instrucțiunea** OPTIMIZE TABLE **pentru tabele supuse la fragmentări.** Tabelele care suferă numeroase modificări, mai ales acelea care conțin coloane cu lungime variabilă, sunt supuse fragmentării. Fragmentarea nu este indicată, deoarece duce la apariția unor spații nedorite în blocurile de pe disc folosite pentru stocarea tabelului dumneavoastră. În timp, trebuie să citiți mai multe blocuri pentru a obține rândurile valide, iar performanțele sunt diminuate. Acest lucru este adevărat pentru orice tabel cu rânduri de lungime variabilă, dar este valabil mai ales pentru coloanele BLOB, deoarece dimensiunea acestora poate varia foarte mult. Utilizarea sistematică a instrucțiunii OPTIMIZE TABLE contribuie la împiedicarea degradării performanțelor tabelului.
- **Folosiți un index sintetic.** Coloanele unui index sintetic pot fi uneori utile. O tehnică este de a crea o valoare hash[3] în funcție de alte coloane și de a o stoca într-o coloană separată. Apoi, puteți găsi rânduri prin căutarea valorilor hash. Acest procedeu este util numai pentru interogările care caută corespondențe exacte cu un model dat. (Valorile hash sunt inutile pentru căutări într-un domeniu cu operatori precum < sau >=.) Valorile hash pot fi generate în MySQL 3.23 sau în versiunile ulterioare folosind funcția MD5().

  Un index hash poate fi deosebit de util în ceea ce privește coloanele BLOB. Unul din motive este că nu puteți indexa aceste tipuri anterior versiunii MySQL 3.23.2. Dar, chiar și în versiunea 3.23.2 sau în versiunile ulterioare, valorile BLOB pot fi mai ușor de găsit folosind o valoare hash ca identificator decât prin căutarea în coloana BLOB.
- **Evitați să regăsiți valori** BLOB **sau** TEXT **prea mari, cu excepția situațiilor când sunteți obligat să o faceți.** De exemplu, o interogare SELECT * nu este o idee prea bună decât dacă sunteți sigur că clauza WHERE va limita rezultatele la rândurile pe care le doriți. În caz contrar, este posibil să deplasați prin rețea valori BLOB posibil foarte mari fără absolut nici un rost. Aceasta este o altă situație când informația de identificare a valorilor BLOB, stocată într-o altă coloană, poate fi utilă. Puteți căuta în coloana respectivă pentru a determina rândul sau rândurile pe care le doriți, după care regăsiți valoarea BLOB din rândurile corespunzătoare.
- **Deplasați valorile** BLOB **într-un tabel separat.** În anumite situații, este justificată mutarea coloanelor BLOB dintr-un tabel în alt tabel secundar, dacă acest lucru vă permite să convertiți tabelul într-un format cu rânduri de lungime fixă pentru coloanele rămase. Acest procedeu va reduce fragmentarea în tabelul primar și, de asemenea, vă permite să profitați de plusul de performanță determinat de rândurile cu lungime fixă.

[3] Valoare numerică rezultată printr-o transformare cunoscută sub numele de funcție hash (în cazul nostru MD5) care convertește un identificator sau o cheie, care au o anumită semnificație pentru utilizator, într-o valoare care indică locul ocupat de date într-un tabel. (Cf. *Dicționar de calculatoare*, Editura Teora, 1999.) – N.T.

Încărcarea eficientă a datelor

Probabil că în majoritatea timpului veți fi preocupat de optimizarea interogărilor SELECT, deoarece acestea sunt cele mai folosite tipuri de interogări și deoarece modul de optimizare a acestora nu este întotdeauna simplu de intuit. Prin comparație, încărcarea datelor în baza dumneavoastră de date este o operație simplă. Totuși, există strategii pe care le puteți folosi pentru a îmbunătăți eficiența operațiunilor de încărcare a datelor. Principiile de bază sunt următoarele:

- Încărcarea unor mari cantități de date este mai rapidă decât încărcarea unui singur rând, deoarece zona cache a indexului nu trebuie golită după încărcarea fiecărei înregistrări; golirea poate avea loc la încheierea lotului de înregistrări.
- Încărcarea este mai rapidă atunci când un tabel nu are indexuri decât atunci când este indexat. Dacă există indexuri, este necesară nu numai adăugarea înregistrării la fișierul de date, dar fiecare index trebuie modificat pentru a reflecta adăugarea noii înregistrări.
- Instrucțiunile SQL mai scurte sunt mai rapide decât instrucțiunile mai lungi, deoarece implică o analiză mai redusă din partea serverului și deoarece pot fi trimise mai rapid prin rețea de la client la server.

Unii din acești factori pot părea minori (mai ales ultimul), dar, dacă încărcați o mulțime de date, chiar și micile trucuri de sporire a eficienței determină o diferență. Putem folosi principiile generale anterioare pentru a trage numeroase concluzii practice cu privire la modul cel mai rapid de încărcare a datelor:

- Instrucțiunea LOAD DATA, în toate formele sale, este mai eficientă decât INSERT, deoarece încarcă rândurile în cantități mari. Golirea zonei cache a indexului are loc mai puțin frecvent și serverul trebuie să analizeze și să interpreteze o singură instrucțiune, în loc de mai multe.
- LOAD DATA este mai eficientă decât LOAD DATA LOCAL. Dacă se folosește instrucțiunea LOAD DATA, fișierul trebuie să se afle în server și dumneavoastră trebuie să aveți privilegiul FILE, dar serverul poate citi fișierul direct de pe disc. Cu instrucțiunea LOAD DATA LOCAL, clientul citește fișierul și îl trimite în rețea serverului, ceea ce reprezintă o operație mai lentă.
- Dacă trebuie să folosiți INSERT, utilizați forma care permite specificarea mai multor rânduri în aceeași instrucțiune:

  ```
  INSERT INTO nume_tabel VALUES(...),(...),...
  ```

  Cu cât se pot specifica numere de rânduri mai mari în cadrul instrucțiunii, cu atât mai bine. Astfel se reduce numărul total de instrucțiuni de care aveți nevoie și se reduce numărul de goliri ale zonei cache a indexului.

  Dacă folosiți mysqldump pentru a genera fișiere copii de siguranță pentru baza de date, folosiți opțiunea --extended-insert, astfel încât fișierul de descărcare să conțină instrucțiuni INSERT pentru mai multe rânduri. De asemenea, puteți folosi --opt (optimizare), care activează opțiunea --extended-insert. Pe de altă parte, evitați utilizarea opțiunii --complete-insert cu mysqldump; instrucțiunile INSERT rezultante vor fi pentru rânduri individuale, vor fi mai lungi și vor necesita un volum suplimentar de analiză decât instrucțiunile generate fără --complete-insert.

- Folosiți protocolul client/server comprimat pentru a reduce cantitatea de date care trec prin rețea. Pentru majoritatea clienților MySQL, acest lucru se poate specifica folosind opțiunea pentru linia de comandă `--compress`. În general, acest protocol se va folosi numai pe rețelele lente, deoarece comprimarea folosește o mare parte din timpul procesorului.
- Lăsați sistemul MySQL să insereze automat valorile prestabilite; nu specificați, în instrucțiunile `INSERT`, coloanele cărora le va fi oricum atribuită valoarea prestabilită. În medie, instrucțiunile dumneavoastră vor fi mai scurte, ceea ce va reduce numărul de caractere trimise prin rețea către server. În plus, deoarece instrucțiunile conțin mai puține valori, serverul execută un volum de analiză mai redus și conversii de valori mai puține.
- Dacă un tabel este indexat, puteți diminua suprasarcina determinată de indexare prin utilizarea inserțiilor grupate pe loturi (instrucțiunea `LOAD DATA` sau instrucțiuni `INSERT` pentru mai multe rânduri). Acestea reduc la minimum impactul actualizării indexului, deoarece zona cache a indexului trebuie golită numai după ce toate rândurile au fost prelucrate, nu după fiecare rând în parte.
- Dacă trebuie să încărcați foarte multe date într-un tabel nou pentru a-l popula, este mai rapid să creați tabelul fără indexuri, să încărcați datele și apoi să creați indexurile. Este mai rapid să creați toate indexurile simultan, în loc de a le modifica pentru fiecare rând în parte.
- Încărcarea datelor într-un tabel indexat poate fi mai rapidă dacă ștergeți sau dezactivați indexurile înainte de încărcare și le reconstruiți sau le reactivați ulterior.

Dacă doriți să folosiți strategia de ștergere sau dezactivare a indexurilor pentru încărcarea datelor, pregătiți-vă să faceți unele experimente, pentru a vedea dacă merită. (Dacă încărcați o cantitate mică de date într-un tabel mare, construirea indexurilor poate lua mai mult timp decât încărcarea datelor.)

Puteți șterge și reconstrui indexuri cu `DROP INDEX` și `CREATE INDEX`. O metodă alternativă este de a dezactiva și reactiva indexurile folosind `myisamchk` sau `isamchk`. Această metodă vă impune să aveți un cont în calculatorul gazdă al serverului MySQL și să aveți acces de scriere în fișierele tabelului. Pentru a dezactiva indexurile unui tabel, treceți la catalogul adecvat al bazei de date și rulați una din următoarele comenzi:

```
% myisamchk --keys-used=0 nume_tabel
% isamchk --keys-used=0 nume_tabel
```

Folosiți `myisamchk` pentru tabelele MyISAM, care au un fișier index cu extensia `.MYI`, respectiv `isamchk` pentru tabelele ISAM, care au un fișier index cu extensia `.ISM`. După încărcarea tabelului cu date, reactivați indexurile:

```
% myisamchk --recover --quick --keys-used=n nume_tabel
% isamchk --recover --quick --keys-used=n nume_tabel
```

`n` este numărul de indexuri pe care îl are tabelul. Puteți determina această valoare prin invocarea utilitarului adecvat cu opțiunea `--description`:

```
% myisamchk --description nume_tabel
% isamchk --description nume_tabel
```


Dacă decideți să folosiți dezactivarea și activarea indexurilor, trebuie să folosiți protocolul de blocare pentru repararea tabelului, descris în capitolul 13, „Întreținerea și repararea bazelor de date", pentru a împiedica serverul să modifice tabelul în același timp cu dumneavoastră. (De fapt nu reparați tabelul, dar îl modificați într-o manieră similară cu procedura de reparare a tabelelor, deci este adecvată folosirea aceluiași protocol.)

Principiile prezentate anterior de încărcare a datelor se aplică și în mediile cu interogări mixte, care implică clienți ce execută diferite categorii de operații. De exemplu, în general doriți să evitați interogări `SELECT` care durează mult pe tabele care sunt actualizate frecvent. Astfel, rezultă o competiție acerbă între clienții care scriu date și o performanță redusă a acestora. O modalitate posibilă de a evita această situație, dacă scrierile dumneavoastră sunt în majoritate operații `INSERT`, este de a adăuga noi înregistrări într-un tabel temporar și apoi de a adăuga periodic acele înregistrări în tabelul principal. Aceasta nu este o strategie viabilă dacă doriți să obțineți acces imediat la noile înregistrări, dar, dacă vă puteți permite să le lăsați inaccesibile pentru o scurtă perioadă de timp, utilizarea tabelului temporar vă va fi de folos din două puncte de vedere. Primul: reduce competiția între interogările `SELECT` care au loc asupra tabelului principal, deci acestea se execută mai rapid. Al doilea: este necesar un timp total mai redus pentru încărcarea unui lot de înregistrări din tabelul temporar în tabelul principal decât cel necesar pentru încărcarea individuală a înregistrărilor; zona cache a indexului trebuie să fie golită numai la finalul fiecărui lot, nu după fiecare rând în parte.

O aplicație a acestei strategii este atunci când consemnați în jurnal numărul de deschideri al paginii Web din serverul dumneavoastră Web într-o bază de date MySQL. În acest caz, probabil că nu este foarte important să vă asigurați că intrările ajung imediat în tabelul principal.

O altă strategie pentru a reduce numărul de goliri ale zonei cache a indexului este de a folosi opțiunea de creare a tabelului `DELAYED_KEY_WRITE` pentru tabelele MyISAM, dacă datele dumneavoastră sunt de așa natură încât nu este absolut esențial ca fiecare înregistrare să fie inserată în cazul unei închideri anormale a sistemului. (Această situație poate apărea dacă folosiți MySQL pentru consemnări în jurnale.) Opțiunea determină golirea ocazională a zonei cache a indexului, nu după fiecare inserție.

Dacă doriți să folosiți golirea întârziată a zonei cache a indexului la nivel de server, lansați utilitarul `mysqld` cu opțiunea `--delayed-key-write`. În acest caz, scrierile în blocul index sunt întârziate până când blocurile trebuie evacuate pentru a face loc altor valori ale indexului, până la execuția unei comenzi `flush-tables` sau până la închiderea tabelului indexat.

Probleme de planificare și de blocare

Secțiunile anterioare s-au concentrat mai ales asupra creșterii vitezei interogărilor individuale. MySQL vă mai permite să modificați prioritățile de planificare a execuției instrucțiunilor, ceea ce poate duce la o mai bună colaborare între interogările care provin de la diferiți clienți, astfel încât clienții individuali să nu rămână blocați pentru o perioadă prea lungă de timp. De asemenea, modificarea priorităților poate garanta faptul că anumite categorii de interogări sunt prelucrate mai rapid. Vom examina mai întâi politica de

planificare prestabilită a sistemului MySQL și apoi vom vedea care sunt opțiunile pe care le aveți la dispoziție pentru influențarea acestei politici. Pentru această expunere, un client care execută o regăsire (o instrucțiune `SELECT`) este un client de citire. Un client care execută o operație care modifică un tabel (instrucțiunile `DELETE`, `INSERT`, `REPLACE` sau `UPDATE`) este un client de scriere.

În esență, politica de planificare din MySQL poate fi rezumată astfel:

- Cererile de scriere trebuie să fie prelucrate în ordinea în care sosesc.
- Scrierile au o prioritate mai mare decât citirile.

Politica de planificare este implementată cu ajutorul blocărilor de tabel. De fiecare dată când un client obține accesul la un tabel, mai întâi este necesară procurarea unei blocări pentru tabelul respectiv. Se poate face acest lucru în mod explicit, cu ajutorul instrucțiunii `LOCK TABLES`, dar în mod normal utilitarul de gestionare a blocărilor din server procură automat blocările, conform necesităților. Când clientul a terminat lucrul cu un tabel, blocarea tabelului poate fi eliminată. O blocare obținută în mod explicit este eliminată cu instrucțiunea `UNLOCK TABLES`, dar și în acest caz serverul eliberează automat blocările pe care le-a obținut.

Un client care efectuează o operație de scriere trebuie să dispună de o blocare pentru acces exclusiv la tabel. Tabelul este într-o stare nedefinită în timpul desfășurării operației, deoarece înregistrările de date sunt șterse, adăugate sau modificate, iar toate indexurile din tabel trebuie actualizate pentru a reflecta modificările. A permite altor clienți să aibă acces la tabel în timp ce acesta se află în flux înseamnă a provoca probleme. Este categoric contraindicat a se permite la doi clienți să scrie în tabel în același timp, deoarece astfel tabelul s-ar deteriora rapid, transformându-se într-o dezordine inutilizabilă. Nu este recomandată nici citirea de către un client a unui tabel aflat în flux, deoarece este posibil ca tabelul să se modifice exact în regiunea în care este citit, ceea ce înseamnă că rezultatele ar fi incorecte.

Un client care efectuează o operație de citire trebuie să dispună de o blocare, pentru a împiedica pe alți clienți să scrie în tabel, astfel încât tabelul să nu se modifice în timp ce este citit. Totuși, blocarea nu trebuie să furnizeze acces exclusiv pentru citire. Blocarea poate permite citirea tabelului de către alți clienți în același timp. Prin citire tabelul nu se modifică, deci nu există nici un motiv pentru care clienții de citire trebuie să se împiedice reciproc să obțină acces la tabel.

MySQL vă permite să influențați politica sa de planificare prin intermediul a numeroși modificatori de interogare. Unul din aceștia este cuvântul cheie `LOW_PRIORITY` (prioritate redusă) pentru instrucțiunile `DELETE`, `INSERT`, `LOAD DATA`, `REPLACE` și `UPDATE`. Un altul este cuvântul cheie `HIGH_PRIORITY` (prioritate ridicată) pentru instrucțiunile `SELECT`, iar cel de-al treilea este cuvântul cheie `DELAYED` (amânat) pentru instrucțiunile `INSERT` și `REPLACE`.

Cuvântul cheie `LOW_PRIORITY` influențează planificarea operațiilor după cum urmează: în mod normal, dacă o operație de scriere într-un tabel ajunge în timp ce tabelul este citit, clientul de scriere se blochează până când clientul de citire își termină execuția deoarece o interogare, o dată pornită, nu poate fi întreruptă. Dacă sosește o altă cerere de citire în timp ce clientul de scriere așteaptă, se va bloca și clientul de citire deoarece, conform

politicii de planificare prestabilite, clienții de scriere au prioritate mai mare decât clienții de citire. Când primul client de citire și-a terminat execuția, clientul de scriere își începe execuția, iar când acesta din urmă termină, al doilea client de citire își începe execuția.

În cazul în care cererea de scriere este o cerere cu prioritate redusă, se consideră că scrierea nu are o prioritate mai ridicată decât citirile. În acest caz, dacă o a doua cerere de citire sosește în timp ce clientul de scriere așteaptă, al doilea client de citire este autorizat să treacă înaintea clientului de scriere. Clientul de scriere are permisiunea de a continua numai atunci când nu mai există clienți de citire. O implicație a acestei modificări în planificarea operațiilor este aceea că, teoretic, este posibil ca scrierile cu prioritate redusă să fie blocate la infinit. Dacă sosesc noi cereri de citire când cererile anterioare sunt încă în curs de desfășurare, noile cereri vor avea permisiunea de a fi executate anterior scrierii cu prioritate redusă.

Cuvântul cheie `HIGH_PRIORITY` pentru interogările `SELECT` este similar; permite unei instrucțiuni `SELECT` să fie executată anterior unei scrieri aflate în așteptare, chiar dacă scrierea are prioritate normală.

Modificatorul `DELAYED` pentru instrucțiunea `INSERT` se comportă astfel: atunci când pentru un tabel sosește o cerere `INSERT DELAYED`, serverul așază rândurile într-o coadă și returnează imediat clientului o stare, astfel încât clientul să-și poată începe execuția chiar înainte de inserția rândurilor. În cazul în care clienții citesc din tabel, rândurile din coadă sunt păstrate. Când nu mai există clienți de citire, serverul începe să insereze rândurile din coada cu rânduri amânate. Ocazional, serverul face o pauză, pentru a vedea dacă au sosit cereri de citire noi și dacă acestea se află în așteptare. În acest caz, coada cu rânduri amânate este suspendată și clienților de citire li se permite să-și înceapă execuția. Când nu au mai rămas clienți de citire, serverul reîncepe să insereze rândurile amânate. Acest proces continuă până când coada se golește.

Modificatorii de planificare nu au apărut în MySQL toți deodată. Tabelul următor prezintă acești modificatori și versiunea programului MySQL în care au apărut. Puteți folosi acest tabel pentru a determina caracteristicile versiunii dumneavoastră de MySQL.

Tipul instrucțiunii	**Versiunea primei apariții**
`DELETE LOW_PRIORITY`	3.22.5
`INSERT LOW_PRIORITY`	3.22.5
`INSERT DELAYED`	3.22.15
`LOAD DATA LOW_PRIORITY`	3.23.0
`LOCK TABLES ... LOW_PRIORITY`	3.22.8
`REPLACE LOW_PRIORITY`	3.22.5
`REPLACE DELAYED`	3.22.15
`SELECT ... HIGH PRIORITY`	3.22.9
`UPDATE LOW PRIORITY`	3.22.5
`SET SQL_LOW_PRIORITY_UPDATES`	3.22.5

Efecte de margine pe parte de client ale instrucțiunii INSERT DELAYED

Instrucțiunea `INSERT DELAYED` este utilă dacă alți clienți rulează instrucțiuni `SELECT` lungi și nu doriți să blocați alți clienți în așteptarea finalizării inserției. Clientul care emite instrucțiunea `INSERT DELAYED` se poate executa mai rapid, deoarece serverul pur și simplu așază la coadă rândul care trebuie inserat.

Totuși, trebuie să cunoașteți și alte diferențe între instrucțiunile `INSERT` normale și comportarea instrucțiunii `INSERT DELAYED`. Clientul primește o eroare dacă instrucțiunea `INSERT` conține o eroare de sintaxă, dar alte informații care erau disponibile în mod normal nu mai sunt acum accesibile. De exemplu, nu veți putea obține valoarea `AUTO_INCREMENT` atunci când instrucțiunea returnează. De asemenea, nu veți obține un număr de duplicate al valorilor din indexurile unice. Aceasta se întâmplă deoarece operația de inserție returnează o stare înainte de finalizarea efectivă a operației. O altă implicație este aceea că, dacă rândurile din instrucțiunile `INSERT DELAYED` sunt puse la coadă în timp ce așteaptă să fie inserate și serverul suferă o cădere sau este suprimat (cu instrucțiunea `kill -9`), rândurile se vor pierde. Acest fapt nu este valabil pentru o suprimare `TERM` normală; în situația respectivă, serverul va insera rândurile înainte de a-și încheia execuția.

Optimizare pentru administratori

Secțiunile anterioare au descris optimizări care pot fi efectuate de către utilizatorii obișnuiți ai sistemului MySQL, privind operațiile de creare și indexare a tabelelor, dar și legate de interogări. Există însă și optimizări care pot fi efectuate numai de către administratorii MySQL și de sistem, care dețin controlul asupra serverului MySQL sau asupra calculatorului pe care rulează acesta. Unii parametri ai serverului au o legătură directă cu prelucrarea interogărilor și pot fi ajustați, după cum anumite aspecte legate de configurarea componentelor hardware au un efect direct asupra vitezei de prelucrare a interogărilor.

Parametrii serverului

Serverul are parametri (variabile) generali pe care îi puteți modifica pentru a-i influența modul de operare. O expunere generală referitoare la ajustarea parametrilor serverului este dată în capitolul 11, „Administrarea generală a sistemului MySQL“, dar o parte din acești parametri sunt corelați mai ales cu prelucrarea interogărilor și merită o scurtă prezentare în acest cadru:

- `delayed_queue_size`

  Acest parametru determină numărul de rânduri din instrucțiunile `INSERT DELAYED` care vor fi stocate în coadă anterior blocării clienților care execută alte instrucțiuni `INSERT DELAYED`. Mărirea acestei valori permite serverului să accepte mai multe rânduri de la acest tip de cerere, astfel încât clienții să se poată executa fără blocare.

- `key_buffer_size`

  Aceasta este dimensiunea bufferului folosit pentru stocarea blocurilor de index. Dacă dispuneți de memoria necesară, mărirea acestei valori trebuie să îmbunătățească intervalul de timp necesar pentru crearea și modificarea indexului. Valori mai ridicate ale acestui parametru permit sistemului MySQL să stocheze în memorie mai multe blocuri de index simultan, ceea ce mărește probabilitatea de găsire a valorilor cheie în memorie, fără a fi necesară citirea unui bloc nou de pe disc.

În MySQL 3.23 și în versiunile ulterioare, dacă măriți dimensiunea bufferului pentru stocarea cheilor, puteți porni serverul folosind opțiunea `--init-file`. Aceasta permite specificarea unui fișier de instrucțiuni SQL, care să fie executate la pornirea serverului. Dacă aveți tabele numai pentru citire pe care doriți să le stocați în memorie, le puteți copia în tabele HEAP, pentru a executa căutări foarte rapide în index.

Aspecte legate de componentele hardware

Pentru îmbunătățirea performanțelor serverului, vă puteți folosi componentele hardware într-o manieră mai eficientă:

- **Instalați o cantitate mai mare de memorie în calculatorul dumneavoastră.** Aceasta vă permite să măriți dimensiunile zonei cache și a bufferelor din memoria serverului. De asemenea, permite serverului să folosească mai frecvent informațiile stocate în memorie și să necesite mai puține operații de preluare a informațiilor de pe disc.
- **Reconfigurați sistemul astfel încât să elimine toate dispozitivele de schimb de memorie cu discul, dacă aveți o cantitate de RAM suficientă pentru a efectua toate operațiile de schimb într-un sistem de fișiere din memorie.** În caz contrar, unele sisteme vor continua să „împrumute“ memorie de pe disc, chiar dacă aveți o cantitate suficientă de memorie RAM pentru operațiile de schimb.
- **Adăugați discuri mai rapide pentru a îmbunătăți latența I/O.** În mod caracteristic, timpul de căutare este elementul hotărâtor pentru performanță în acest caz. Deplasarea capetelor de citire în lateral este o operație lentă; odată capetele poziționate, citirea blocurilor din afara pistei este, comparativ, mai rapidă.
- **Încercați să redistribuiți activitatea discului pe dispozitive fizice.** De exemplu, dacă vă puteți stoca două dintre cele mai solicitate baze de date ale dumneavoastră pe unități fizice separate, procedați în acest sens. Rețineți că utilizarea a diferite partiții pe același dispozitiv fizic nu este suficientă. Acest lucru nu vă va fi de folos, deoarece partițiile vor continua să concureze pentru aceeași resursă fizică (în speță capetele discului). Procedura pentru mutarea bazelor de date este descrisă în capitolul 10, „Catalogul de date MySQL“.

  Înainte de a muta datele într-un alt dispozitiv, asigurați-vă că înțelegeți caracteristicile de încărcare ale sistemului dumneavoastră. Dacă o altă activitate importantă este deja în curs de desfășurare pe un anumit dispozitiv fizic, inserția unei baze de date în acel dispozitiv nu ar face decât să înrăutățească performanțele. De exemplu, nu veți obține nici un avantaj global dacă prelucrați o mare cantitate de trafic Web și mutați o bază de date în dispozitivul unde este localizat arborele document al serverului dumneavoastră Web. (Dacă dispuneți numai de o singură unitate, nu aveți cum să procedați la o redistribuire a activității discului, desigur.)
- **Când construiți MySQL, configurați-l astfel încât să folosească biblioteci statice, nu biblioteci partajate.** Fișierele binare dinamice care folosesc bibliotecile partajate determină economii de spațiu pe disc, dar fișierele binare statice sunt mai rapide. (Totuși, nu puteți folosi fișiere binare statice dacă doriți să încărcați funcții definite de utilizator, deoarece mecanismul UDF se bazează pe legături dinamice.)

PARTEA A-II A

Utilizarea interfețelor de programare ale sistemului MySQL

CAPITOLUL 5

Introducere în programarea MySQL

În această parte a cărții, vom discuta despre ceea ce trebuie să știți pentru a vă scrie propriile dumneavoastră programe care obțin accesul la bazele de date MySQL. MySQL vine cu un set de programe utilitare. De exemplu, `mysqldump` exportă conținutul și definițiile structurii tabelelor, `msqlimport` încarcă fișierele de date în tabele, `mysqladmin` execută operații administrative, iar `mysql` vă permite să interacționați cu serverul pentru a executa interogări arbitrare. Fiecare din utilitarele standard din MySQL este conceput ca un program mic, cu o anumită finalitate și cu o funcție specifică, limitată. Acest lucru este adevărat chiar și pentru `mysql`, care este mai flexibil decât celelalte utilitare, în sensul că îl puteți folosi pentru a executa orice număr de interogări diferite: este conceput cu scopul unic de a vă permite să emiteți instrucțiuni SQL direct către server și să vizualizați rezultatele.

Această natură limitată a clienților MySQL nu este un neajuns, ci derivă din proiectare. Programele sunt utilitare de uz general; nu sunt menite a anticipa toate posibilele cerințe pe care le puteți avea. Dezvoltatorii sistemului MySQL nu sunt adepți ai principiului de a scrie programe imense, de dimensiuni excesive, care încearcă să facă orice ați putea dori să faceți (sfârșind astfel prin a include mari cantități de program pentru lucruri care nu vă interesează câtuși de puțin). Cu toate acestea, uneori aplicațiile au cerințe care nu pot fi rezolvate prin funcționalitățile programelor client standard. Pentru a aborda aceste cazuri, MySQL furnizează o bibliotecă de programare a clienților. Aceasta vă permite să vă scrieți propriile dumneavoastră programe și vă oferă flexibilitatea de a satisface toate cerințele specializate pe care aplicațiile dumneavoastră le-ar putea avea. Oferindu-vă acces la serverul MySQL, biblioteca client deschide posibilități limitate numai de propria dumneavoastră imaginație.

Care sunt acele funcționalități specifice pe care le puteți acumula scriindu-vă propriile programe? Să examinăm această întrebare comparativ cu funcționalitățile clientului `mysql` și a interfeței sale simple cu serverul MySQL:

- **Puteți personaliza metodele de manipulare a datelor de intrare.** Cu `mysql`, introduceți instrucțiuni SQL brute. Cu propriile dumneavoastră programe, puteți furniza utilizatorului metode de introducere a datelor mai intuitive și mai ușor de folosit. Programul poate elimina necesitatea ca utilizatorul să cunoască limbajul SQL și chiar pe aceea de a cunoaște rolul bazei de date în operația care se execută.

  Colecția de date de intrare poate fi ceva rudimentar, ca un ciclu de citire a promptului și a valorilor introduse într-o interfață în stil linie de comandă, respectiv ceva sofisticat, ca un formular de introducere a datelor bazat pe ecran, implementat folosind un pachet de gestiune a ecranului precum `curses` sau S-Lang, un sistem X (window) care folosește Tcl/Tk sau un formular dintr-un browser Web.

  Pentru majoritatea utilizatorilor, specificarea parametrilor de căutare este cu mult mai simplă prin completarea unui formular decât prin emiterea unei instrucțiuni `SELECT`. De exemplu, un agent imobiliar care caută case care se încadrează într-un anumit domeniu de preț, construite într-un anumit stil sau amplasate într-o anumită locație nu dorește decât să introducă parametrii de căutare într-un formular și să primească ofertele corespunzătoare cu un minimum de deranj. Pentru introducerea de noi înregistrări sau actualizarea înregistrărilor existente, sunt valabile considerații similare. O dactilografă dintr-un departament de introducere a datelor nu trebuie să cunoască sintaxa SQL pentru instrucțiunile `INSERT`, `REPLACE` sau `UPDATE`.

  Un motiv suplimentar pentru interpunerea unui nivel de colectare a datelor de intrare între utilizatorul final și serverul MySQL este acela că puteți valida datele de intrare introduse de utilizator. De exemplu, puteți verifica datele, pentru a vă asigura că acestea corespund formatului așteptat de MySQL, sau puteți impune completarea anumitor câmpuri.

- **Puteți personaliza datele de ieșire.** În esență, datele de ieșire ale programului `mysql` sunt neformatate; puteți alege un stil tabular (sau delimitat prin tabulatori). Dacă doriți date de ieșire într-un format mai estetic, trebuie să le formatați dumneavoastră personal. Această formatare variază între operații simple (cum ar fi a scrie „Inexistent" în loc de `NULL`) și până la cerințe mai complexe de generare a rapoartelor. Să luăm în considerare următorul raport:

```
Stat  Oras                    Vanzari
-------------------------------------
AZ    Mesa               94384,24 USD
      Phoenix            17328,28 USD
      -------------------------------
      subtotal          117712,52 USD
-------------------------------------
CA    Los Angeles       118198,18 USD
      Oakland            38838,36 USD
      -------------------------------
      Subtotal          157036,54 USD
=====================================
      TOTAL GENERAL     274749,06 USD
```

Acest raport include numeroase elemente specializate:

- Antete personalizate.
- Eliminarea valorilor care se repetă în coloana `Stat`, astfel încât valorile să fie afișate numai atunci când se modifică.
- Calcule de subtotal și total general.
- Formatarea numerelor, cum este `94384,24`, ca valori exprimate în dolari, adică `94384,24 USD`.

Pentru unele categorii de sarcini, nici măcar nu aveți nevoie de date de ieșire. Poate că pur și simplu regăsiți informații pentru a calcula un rezultat pe care îl inserați într-un alt tabel din baza de date. Poate chiar doriți ca datele de ieșire să ajungă în altă parte

decât la utilizatorul care rulează interogarea. De exemplu, dacă extrageți nume și adrese de e-mail pentru a alimenta în mod automat un proces care generează scrisori tip pentru mesaje de e-mail în cantități mari, programul dumneavoastră produce date de ieșire. Dar datele de ieșire sunt alcătuite din mesaje care se deplasează la destinatarii mesajelor de e-mail, nu la persoana care rulează programul.

- **Puteți evita restricțiile impuse de natura intrinsecă a limbajului SQL.** SQL nu este un limbaj procedural, cu un set de structuri de control al fluxului, cum sunt condiționalele, ciclurile sau subrutinele. Scripturile SQL sunt compuse dintr-un set de instrucțiuni care se execută una câte una, de la început la sfârșit, cu o minimă verificare a apariției erorilor.

  Dacă executați un fișier cu interogări SQL folosind `mysql` în modul batch, `mysql` fie va renunța după prima eroare, fie – dacă specificați opțiunea `--force` – execută toate interogările fără nici o discriminare, indiferent de numărul de erori care se produc. Un program poate asigura controlul fluxului în jurul instrucțiunilor, astfel încât dumneavoastră să vă puteți adapta în mod selectiv la starea de succes sau de eșec a interogărilor. Puteți face execuția unei interogări independentă de reușita sau eșecul alteia sau puteți lua decizii cu privire la următoarea acțiune în funcție de rezultatul unei interogări anterioare.

  SQL are o persistență foarte redusă de-a lungul instrucțiunilor, fapt care se manifestă și la `mysql`. Este dificilă utilizarea rezultatelor unei interogări și aplicarea lor într-o altă interogare, respectiv coroborarea rezultatelor mai multor interogări. Funcția `LAST_INSERT_ID()` se poate folosi pentru a obține valoarea `AUTO_INCREMENT` cel mai recent generată de o instrucțiune anterioară, dar cam asta e tot...

  Mai general, se poate dovedi dificilă regăsirea unui set de înregistrări, urmată de utilizarea fiecăreia ca bază pentru o serie complexă de alte operații. De exemplu, regăsirea unei liste de clienți și apoi căutarea într-un istoric detaliat al creditului pentru fiecare pot implica mai multe interogări pentru fiecare client. În anumite situații, doriți să produceți o factură pentru care trebuie să asociați antetul facturii cu informațiile despre client și despre fiecare articol comandat. `mysql` nu este adecvat pentru aceste categorii de operații, atât deoarece aveți nevoie de mai multe interogări care depind de rezultatele unor interogări anterioare, precum și fiindcă aceste operații depășesc posibilitățile de formatare ale programului `mysql`.

  În general, pentru operații care implică relații master-detaliu și care prezintă cerințe complexe de formatare a datelor de ieșire este nevoie de un alt instrument decât `mysql`. Un program asigură liantul care corelează interogările și care vă permite să folosiți datele de ieșire ale unei interogări ca date de intrare pentru alta.

- **Puteți integra MySQL în orice aplicație.** Multe programe își extrag avantajele din capacitatea de a exploata capacitatea unui sistem de baze de date de a furniza informații. O aplicație care trebuie să verifice un număr de client sau să afle dacă un anumit articol se găsește sau nu în inventar poate efectua această operație prin emiterea unei interogări rapide. O aplicație Web care permite unui client să solicite toate cărțile scrise de un anumit autor le poate căuta pe acestea într-o bază de date și apoi prezintă rezultatele browserului clientului.

  Este posibilă realizarea unui grad rudimentar de „integrare" prin utilizarea unui script de interpretor care invocă `mysql` cu un fișier de intrare care conține instrucțiuni SQL, iar apoi prin postprocesarea datelor de ieșire folosind alte utilitare UNIX. Acest procedeu poate deveni neatractiv, pe măsură ce operația dumneavoastră progresează. De asemenea, poate da o senzație de genul „merge, dar ceva nu-i în regulă" pe măsură ce aplicația evoluează într-o cârpeală dezordonată. De asemenea, suprasarcina de creare de proces a unui script de interpretor care rulează alte comenzi poate fi mai mare decât doriți dumneavoastră să o creați. Poate fi mai eficient să interacționați direct cu serverul MySQL, extrăgând exact informațiile de care aveți nevoie pe măsura necesităților, în fiecare fază de execuție a aplicației dumneavoastră.

În ceea ce privește baza noastră de date demonstrativă `samp_db` pe care am configurat-o în capitolul 1, „Introducere în MySQL și SQL", am enumerat acolo numeroase deziderate care ne impun să scriem programe care să interacționeze cu serverul MySQL. Unele din aceste deziderate sunt prezentate în următoarea listă:

- Formatarea catalogului membrilor Ligii istorice în vederea tipăririi
- Facilitarea prezentării catalogului și a căutării în acesta din Internet
- Trimiterea prin e-mail a înștiințărilor de plată a cotizației
- Introducerea cu ușurință a punctajelor în catalogul școlar folosind un browser Web

Un domeniu pe care îl vom trata în detaliu îl constituie integrarea funcționalităților sistemului MySQL într-un mediu Web. MySQL nu furnizează suport direct pentru aplicații Web dar, prin combinarea acestuia cu instrumentele adecvate, bazele dumneavoastră de date vor deveni ușor accesibile prin Web. Puteți specifica interogări folosind serverul dumneavoastră de Web și puteți raporta rezultatele browserului unui client.

Există două perspective complementare ale „căsniciei" între MySQL și Web:

- Interesul dumneavoastră primordial este baza de date, iar dumneavoastră doriți să folosiți Web-ul ca instrument pentru obținerea unui acces mai ușor la date. Locul unui sistem de baze de date într-un atare scenariu este explicit și evident, deoarece este punctul focal al interesului dumneavoastră. De exemplu, puteți scrie pagini Web care vă permit să vedeți care sunt tabelele pe care le conține baza dumneavoastră de date, care este structura fiecărui tabel și care este conținutul acestuia. Serverul Web îl folosiți pentru a îmbunătăți accesul la MySQL. Acesta este punctul de vedere probabil al unui administrator MySQL.
- Interesul dumneavoastră fundamental poate fi situl Web, și puteți dori să folosiți MySQL ca instrument pentru a spori valoarea conținutului sitului dumneavoastră pentru persoanele care îl vizitează. De exemplu, dacă rulați un avizier electronic sau o listă de discuții pentru vizitatorii sitului, puteți folosi MySQL pentru a ține evidența mesajelor. În acest caz, rolul sistemului de baze de date este mai subtil și este posibil chiar ca vizitatorii să nu-și dea seama că acesta este angrenat în serviciile pe care le oferiți. Folosiți MySQL pentru a extinde funcționalitățile serverului dumneavoastră de Web. Acesta este, probabil, punctul de vedere al dezvoltatorului unui sit Web.

Aceste perspective nu sunt mutual exclusive. De exemplu, în scenariul care implică Liga istorică, dorim să folosim Web-ul ca mijloc pentru ca membrii să obțină un acces rapid la conținutul catalogului cu membri, punând rubricile catalogului la dispoziție prin Internet.

Aceasta constituie o utilizare a Web-ului pentru furnizarea accesului la baza de date. În acelaşi timp, situl Web al Ligii este oarecum „subdezvoltat“, deci inserţia în cadrul sitului a conţinutului catalogului măreşte valoarea sitului în ochii membrilor. Aceasta este o utilizare a sistemului de baze de date pentru extinderea serviciilor furnizate de sit.

Indiferent de modul în care vedeţi integrarea sistemului MySQL cu Web-ul, implementarea este similară. Conectaţi interfaţa sitului dumneavoastră Web la componenta back end MySQL, folosind serverul Web ca intermediar. Serverul Web trimite o interogare de la utilizator către serverul MySQL, regăseşte rezultatele interogării şi apoi le transmite clientului în vederea vizualizării într-un browser.

Nu doriţi să vă puneţi datele în reţea, desigur, dar uneori această operaţie îşi are avantajele sale, mai ales în comparaţie cu accesul la datele dumneavoastră prin intermediul programelor client MySQL standard:

- Persoanele care obţin accesul la datele dumneavoastră prin intermediul Web-ului pot folosi browserul pe care îl preferă, pe tipul de platformă pe care îl preferă. Nimeni nu este obligat să folosească numai sistemele pe care pot rula programele client MySQL. Indiferent de aria de acoperire a clienţilor MySQL, browserele Web sunt şi mai răspândite.
- Utilizarea unei interfeţe Web poate deveni mai simplă decât un client autonom MySQL în linie de comandă.
- O interfaţă Web poate fi personalizată conform cerinţelor unei anumite aplicaţii. Clienţii MySQL sunt instrumente de uz general, cu interfaţă fixă.
- Paginile Web dinamice extind posibilităţile sistemului MySQL de a efectua operaţii care sunt dificil sau imposibil de realizat folosind clienţii MySQL. De exemplu, nu puteţi asambla o aplicaţie care încorporează un „cărucior de cumpărături“ electronic folosind numai clienţi MySQL.

Pentru scrierea aplicaţiilor bazate pe Web se poate folosi orice limbaj de programare, dar unele sunt mai adecvate decât altele. Vom vedea acest lucru în secţiunea „Alegerea unei interfeţe de programare a aplicaţiilor“.

Interfeţe de programare a aplicaţiilor (API) disponibile pentru MySQL

Pentru facilitarea dezvoltării aplicaţiilor, MySQL furnizează o bibliotecă client, scrisă în limbajul de programare C, care vă permite să obţineţi accesul la bazele de date MySQL din interiorul oricărui program scris în C. Biblioteca client implementează o interfaţă de programare a aplicaţiilor (API) care defineşte modul în care programele client stabilesc şi derulează comunicaţiile cu serverul.

Totuşi, nu sunteţi obligat să folosiţi numai limbajul C pentru a scrie programe MySQL. Multe alte procesoare de limbaj sunt fie scrise în C, fie au capacitatea de a folosi biblioteci C, deci biblioteca client MySQL furnizează mijloacele prin care asocierile MySQL pentru aceste limbaje pot fi construite pe baza interfeţei API pentru C. Această facilitate vă oferă numeroase opţiuni pentru scrierea aplicaţiilor care „discută“ cu serverul MySQL. Interfeţe API client există pentru Perl, PHP, Java, Python, C++, Tcl şi altele.

Căutaţi în manualul de referinţă MySQL sau în situl Web MySQL o listă actualizată, deoarece ocazional sunt adăugate noi interfeţe API de limbaj.

Fiecare asociere de limbaj îşi defineşte propria sa interfaţă, care specifică regulile de acces la MySQL. Spaţiul nu ne permite să discutăm despre fiecare din interfeţele API disponibile pentru MySQL, deci ne vom concentra asupra a trei dintre cele mai populare:

- **Interfaţa API a bibliotecii client în C.** Aceasta este principala interfaţă de programare cu MySQL.
- **Interfaţa API DBI (Database Interface) pentru limbajul de scripting de uz general Perl.** DBI este implementată ca un modul Perl, care interfaţează cu alte module la nivelul DBD (Database Driver), fiecare modul furnizând acces la un anumit tip de motor pentru baze de date. (Modulul DBD particular asupra căruia ne vom concentra este cel care furnizează suport pentru MySQL, desigur.) Cele mai comune utilizări ale DBI pentru MySQL sunt scrierea de clienţi autonomi care sunt invocaţi de la linia de comandă, precum şi de scripturi destinate a fi invocate de un server Web pentru a furniza acces din Web la MySQL.
- **Interfaţa API pentru PHP.** PHP este un limbaj de scripting care furnizează o modalitate convenabilă de înglobare a programelor în paginile Web. O asemenea pagină este prelucrată de PHP înainte de a fi trimisă clientului, ceea ce permite scriptului să genereze un conţinut dinamic, cum ar fi includerea în pagină a rezultatului unei interogări MySQL. Iniţial, PHP avea semnificaţia "Personal Home Page" (pagină de bază personală), dar acest limbaj s-a dezvoltat mult dincolo de umilele sale începuturi. Situl Web PHP foloseşte acum numele în contextul "PHP: Hypertext Preprocessor", care face referire la sine însuşi în aceeaşi manieră ca şi GNU ("GNU's Not UNIX") (GNU nu este UNIX).

Construiţi pornind de la rezultatele muncii altora

Când clienţii MySQL standard sunt insuficienţi pentru necesităţile dumneavoastră, nu este necesar întotdeauna să vă scrieţi propriile programe. Alţii s-au ocupat deja cu scrierea unor programe, din care multe sunt disponibile gratuit. Vezi Anexa I, „Instrumente utile produse de terţe părţi“, pentru unele exemple. Este posibil să găsiţi câteva care să vă scutească de ceva muncă.

Fiecare din aceste trei interfeţe API este prezentată în detaliu în propriul său capitol. Capitolul de faţă conţine o trecere în revistă comparativă a interfeţelor API, pentru a descrie caracteristicile generale ale acestora şi pentru a vă oferi o idee privind alegerea uneia în dauna alteia pentru o anumită aplicaţie.

Desigur, nu aveţi nici un motiv să vă consideraţi „înţepenit“ într-o singură interfaţă API. Învăţaţi să cunoaşteţi fiecare interfaţă şi înarmaţi-vă cu acele cunoştinţe care vă permit să faceţi o alegere înţeleaptă. Dacă aveţi un proiect de mari dimensiuni, cu numeroase componente, puteţi folosi mai multe interfeţe API şi puteţi scrie unele componente într-un limbaj şi alte componente într-un alt limbaj, în funcţie de alegerea cea mai adecvată pentru fiecare parte a proiectului. De asemenea, este instructiv să implementaţi o aplicaţie în mai multe moduri, dacă timpul vă permite. Astfel, căpătaţi o experienţă directă în utilizarea diferitelor interfeţe API, pe măsură ce acestea se aplică propriilor dumneavoastră aplicaţii.

Dacă trebuie să vă procurați programele necesare pentru utilizarea oricăreia dintre interfețele API pe care doriți să le încercați, consultați Anexa A, „Obținerea și instalarea programelor", pentru a primi instrucțiuni.

Interfața API pentru C

Interfața API pentru C este folosită în contextul programelor C compilate. Aceasta este o bibliotecă client care furnizează interfața de nivelul cel mai redus disponibilă pentru comunicarea cu serverul MySQL, oferindu-vă funcționalitățile de care aveți nevoie pentru stabilirea unei conexiuni cu serverul și conversația cu acesta.

Predecesorii DBI și PHP

Predecesorul Perl al DBI este modulul Mysqlperl, și anume `Mysql.pm`. Acest modul nu mai este acceptat și nu trebuie folosit pentru dezvoltarea noilor programe MySQL. Un motiv este acela că Mysqlperl este dependent de MySQL, în timp ce DBI nu este. Dacă scrieți aplicații Perl pentru MySQL și apoi vă decideți că doriți să le folosiți cu un alt motor pentru baze de date, portarea scripturilor DBI este mai simplă decât aceea a scripturilor Mysqlperl, deoarece primele sunt mai puțin dependente de un anumit motor de baze de date.

Dacă vă procurați un script Perl pentru accesul la MySQL și descoperiți că, în loc de a fi scris pentru DBI, este scris pentru Mysqlperl, puteți totuși să folosiți DBI. DBI poate fi construit astfel încât să includă suport de emulare pentru Mysqlperl, deci nu este necesar să instalați ambele pachete.

Predecesorul lui PHP 3 este PHP/FI 2.0 (FI este abrevierea de la *form interpreter* - interpretor de formulare). Ca și Mysqlperl, PHP/FI este depășit și nu vom mai discuta despre el.

Originea interfeței API în C cu MySQL

Dacă aveți experiență în scrierea de programe pentru sistemul de gestiune a bazelor de date relaționale mSQL, veți observa că interfața API în C pentru MySQL este similară cu interfața API în C corespunzătoare pentru mSQL. Când dezvoltatorii MySQL au început să-și implementeze motorul SQL propriu, pentru mSQL erau disponibile un număr de utilitare gratuite. Pentru a face posibilă portarea acelor utilitare mSQL către MySQL cu minimum de dificultate, interfața API MySQL a fost proiectată intenționat astfel încât să fie similară cu interfața API pentru mSQL. (MySQL este chiar dotat cu un script `msql2mysql`, care execută substituții textuale simple al numelor de funcții din interfața API pentru mSQL în numele MySQL corespunzătoare. Această operație este relativ simplă, dar execută o bună parte din activitatea impusă de conversia unui program mSQL în vederea utilizării cu MySQL.)

Clienții C furnizați în distribuția MySQL sunt bazați pe această interfață API. Biblioteca client C mai servește și drept bază pentru asocierile MySQL cu alte limbaje, cu excepția interfețelor API pentru Java. De exemplu, driverul specific MySQL pentru modulul Perl DBI și programul PHP au devenit ambele compatibile cu MySQL prin legarea codului pentru biblioteca client în C pentru MySQL. (Acest proces este ilustrat în instrucțiunile de instalare a modulelor DBI și PHP din Anexa A.)

Interfața API pentru modulul Perl DBI

Interfața API pentru DBI este folosită în contextul aplicațiilor scrise pentru limbajul de scripting Perl. Aceasta este interfața care conține arhitectura pe cele mai multe etaje dintre cele trei interfețe API despre care vom discuta, deoarece încearcă să lucreze cu un număr

de sisteme de baze de date cât mai mare posibil, ascunzând în același timp generatorului de scripturi cât mai multe detalii specifice.

DBI este implementat prin intermediul modulelor Perl care folosesc o arhitectură cu două niveluri (vezi figura 5.1):

- **Nivelul DBI (interfață cu baza de date).** Furnizează interfața pentru scripturi client. Acest nivel furnizează o abstractizare care nu face referire la motoare pentru baze de date specifice.
- **Nivelul DBD (driver pentru baze de date).** La acest nivel este furnizat un suport pentru diferite motoare de baze de date, de către drivere care sunt specifice fiecărui motor de baze de date.

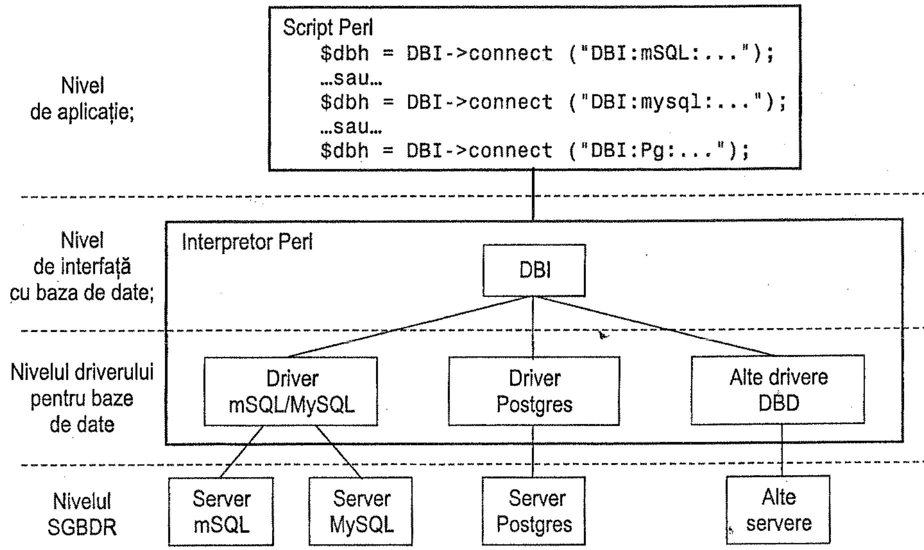


Figura 5.1 Arhitectura DBI.

Suportul MySQL pentru DBI este furnizat de către distribuția `Msql-Mysql-modules`. Acest modul operează la nivelul DBD. După cum puteți intui din numele distribuției și de asemenea din figura 5.1, este posibil ca un driver să furnizeze suport pentru mai multe sisteme de gestiune a bazelor de date relaționale. Inițial, distribuția `Msql-Mysql-modules` fusese scrisă pentru mSQL, apoi extinsă ulterior și pentru MySQL. Acest fapt reflectă asemănările dintre interfețele API scrise în C pentru mSQL și MySQL. Din moment ce interfața API scrisă în C pentru MySQL a fost proiectată asemănător cu interfața API în C pentru mSQL, era logic să se extindă driverul pentru baze de date mSQL (care folosește interfața API pentru mSQL scrisă în C) astfel încât acesta să lucreze cu MySQL.

Arhitectura DBI vă permite să scrieți aplicații într-o manieră relativ generală. Când scrieți un script DBI, folosiți un set standard de apeluri. Stratul DBI invocă driverul adecvat de la nivelul DBD pentru rezolvarea cerințelor dumneavoastră, iar driverul se ocupă de aspectele specifice ale comunicării cu serverul specific de baze de date pe care

doriţi să-l folosiţi. Nivelul DBD transmite datele returnate de server înapoi către nivelul DBI, care prezintă datele aplicaţiei dumneavoastră. Forma datelor este consecventă, indiferent de sistemul de baze de date din care au provenit datele.

Rezultatul este o interfaţă care, din punctul de vedere al creatorului aplicaţiei, ascunde diferenţele între motoarele de baze de date, dar lucrează cu o mare varietate de motoare, cu un număr egal cu acela al driverelor. DBI asigură o interfaţă consecventă cu clientul, care măreşte portabilitatea permiţându-vă să obţineţi accesul la fiecare bază de date într-o manieră uniformă.

Singurul aspect al redactării scripturilor care este în mod obligatoriu specific bazei de date apare la deschiderea unei baze de date. Când stabiliţi conexiunea, indicaţi driverul pe care îl folosiţi. De exemplu, pentru a folosi o bază de date MySQL, vă conectaţi astfel:

```
$dbh = DBI->connect ("DBI:mysql:...");
```

Pentru a folosi în schimb Postgres sau mSQL, vă conectaţi astfel:

```
$dbh = DBI->connect ("DBI:Pg:...");
$dbh = DBI->connect ("DBI:mSQL:...");
```

După ce aţi stabilit conexiunea, nu trebuie să mai faceţi nici o referire specifică la driver. Lăsaţi rezolvarea detaliilor specifice bazelor de date în seama modulului DBI şi a driverului.

Aşa stau lucrurile, cel puţin teoretic. Totuşi, există cel puţin doi factori care subminează portabilitatea scripturilor DBI:

- Implementările SQL diferă între motoarele SGBDR şi este perfect posibil să scrieţi pentru un motor programe SQL pe care alt motor nu le poate înţelege. Dacă programul dumneavoastră SQL este suficient de general, scripturile dumneavoastră vor fi, în mod corespunzător, portabile de la un motor la altul. Dacă programul dumneavoastră SQL este dependent de motor, scripturile vor fi de asemenea dependente de motor. De exemplu, dacă folosiţi instrucţiunea `SHOW TABLES` specifică sistemului MySQL, scriptul dumneavoastră nu va funcţiona cu alte sisteme de baze de date.
- Modulele DBD furnizează deseori tipuri de informaţii specifice diferitelor motoare, pentru a permite creatorilor de scripturi să folosească anumite caracteristici ale sistemelor de baze de date. De exemplu, modulul DBD pentru MySQL asigură o modalitate de acces la proprietăţile coloanelor prin rezultatul unei interogări, precum lungimea maximă a fiecărei coloane, caracterul numeric sau ne-numeric al coloanelor etc. Aceste proprietăţi nu trebuie să aibă vreun analog în alte sisteme de baze de date. Caracteristicile specifice modulului DBD sunt antitetice în raport cu portabilitatea, iar prin utilizarea lor este îngreunată utilizarea în alte sisteme de baze de date a unui script scris pentru MySQL[1].

În ciuda potenţialului acestor doi factori de a face din scripturile dumneavoastră elemente specifice sistemelor de baze de date, mecanismul DBI pentru furnizarea accesului la bazele de date într-o manieră abstractă reprezintă un mijloc rezonabil de realizare a portabilităţii. Este la latitudinea dumneavoastră să decideţi în ce măsură doriţi să beneficiaţi de acest avantaj.

---

[1] Cu toate acestea, veţi descoperi că în capitolul 7, „Interfaţa API pentru Perl DBI", nu se fac prea multe eforturi pentru a evita construcţiile specifice sistemului MySQL furnizate de modulul DBD pentru MySQL. Aceasta deoarece trebuie să ştiţi ce anume reprezintă construcţiile respective, pentru a decide personal dacă le veţi folosi sau nu. – N.A.

Interfaţa API pentru PHP

Ca şi Perl, PHP este un limbaj de scripting. Spre deosebire de Perl, PHP este conceput în mai mică măsură ca limbaj de uz general decât ca limbaj pentru scrierea aplicaţiilor Web. Interfaţa API pentru PHP este folosită mai ales ca mijloc pentru înglobarea scripturilor executabile în paginile Web. Acest fapt facilitează dezvoltatorilor Web scrierea de pagini cu un conţinut generat dinamic. Când un browser client trimite o cerere de pagină PHP unui server Web, PHP execută toate scripturile pe care le găseşte în pagină şi le înlocuieşte cu datele de ieşire ale acestora. Rezultatul este trimis browserului. Din acest motiv, pagina care apare efectiv în browser diferă în funcţie de circumstanţele în care a fost solicitată. De exemplu, când următorul script PHP scurt este înglobat într-o pagină Web, afişează adresa IP a gazdei care a solicitat pagina:

```
<?php echo $REMOTE_ADDR; ?>
```

Ca o aplicaţie mai interesantă şi mai puţin banală, puteţi folosi un script pentru a furniza vizitatorilor informaţii de ultim moment, bazate pe conţinutul bazei dumneavoastră de date. Exemplul următor prezintă un script simplu, cum este cel care ar putea fi folosit în situl Web al Ligii istorice. Scriptul emite o interogare pentru a determina numărul curent de membri ai Ligii şi îl raportează persoanei care vizitează situl (În cazul producerii unei erori, scriptul pur şi simplu nu raportează nici un număr.):

```
<HTML>
<HEAD>
<TITLE>Liga istorica americana</TITLE>
</HEAD>
<BODY>
```

Semnificaţia abrevierilor DBI şi DBD

Deşi nivelul DBI este independent de sistemul de baze de date, iar nivelul DBD este dependent de acest sistem, nu aceasta este semnificaţia abrevierilor DBI, respectiv DBD[2]. Sensul lor este "*database interface*" (interfaţă cu baza de date), respectiv "*database driver*" (driver pentru baze de date).

```
<P>Bine ati venit la situl Web al Ligii istorice.
<?php
$link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
    or exit();
mysql_select_db ("samp_db")
    or exit();
$result = mysql_query ("SELECT COUNT(*) FROM membru")
    or exit();
if ($row = mysql_fetch_array ($result))
    echo "<P>In prezent, Liga are " . $row[0] . " membri";
mysql_free_result ($result);
?>
</BODY></HTML>
```

---

[2] În sensul că DBI se mai poate abrevia (eronat) sub forma *database independent* (independent de sistemul de baze de date), iar DBD se poate interpreta (tot eronat) sub forma *database dependent* (dependent de sistemul de baze de date) – N.T.

În mod caracteristic, scripturile PHP arată ca pagini HTML cu scripturi înglobate între etichetele <?php şi ?>. O pagină poate conţine mai multe scripturi. Acest procedeu furnizează o abordare extrem de flexibilă pentru dezvoltarea scripturilor. De exemplu, dacă doriţi, puteţi scrie o pagină HTML normală pentru a construi cadrul general al paginii, după care adăugaţi mai târziu conţinutul de scripturi al paginii.

PHP nu face nici un efort de a unifica interfaţa cu diferite motoare de baze de date, aşa cum procedează DBI. În schimb, interfaţa cu fiecare motor seamănă foarte mult cu interfaţa pentru biblioteca C corespunzătoare care implementează interfaţa API de nivel scăzut pentru motorul respectiv. De exemplu, numele funcţiilor PHP pe care le folosiţi pentru a obţine acces la MySQL din interiorul scripturilor PHP sunt foarte asemănătoare cu numele funcţiilor din biblioteca client C pentru MySQL.

Alegerea unei interfeţe de programare a aplicaţiilor

Această secţiune conţine îndrumări de ordin general, pentru a vă ajuta să alegeţi o interfaţă API pentru diferitele tipuri de aplicaţii. Sunt comparate funcţionalităţile interfeţelor API pentru C, DBI şi PHP, pentru a vă oferi o idee cu privire la avantajele şi dezavantajele lor relative, precum şi pentru a indica momentul oportun al alegerii uneia sau a alteia dintre interfeţe.

Probabil că ar trebui să arăt, mai întâi, că nu pledez pentru utilizarea vreunuia dintre aceste limbaje în detrimentul altora. Le folosesc pe toate, deşi am preferinţele mele. Şi dumneavoastră veţi avea propriile preferinţe, aşa cum le au şi cei care au examinat manuscrisul acestei cărţi. De fapt, unul dintre recenzori a fost de părere că ar trebui să pun un accent mai mare pe importanţa limbajului C pentru programarea în MySQL, în timp ce altul credea că ar fi cazul să critic zdravăn programarea în C şi să descurajez utilizarea acestui limbaj! Va trebui să cântăriţi factorii discutaţi în această secţiune şi să ajungeţi la propriile dumneavoastră concluzii.

În alegerea dumneavoastră privind interfaţa API de selectat pentru îndeplinirea unei anumite sarcini trebuie să ţineţi cont de un număr de consideraţii:

- **Mediul de execuţie scontat.** Contextul în care vă aşteptaţi să fie folosită aplicaţia.
- **Performanţe.** Eficienţa derulării aplicaţiilor atunci când sunt scrise în limbajul interfeţei API.
- **Simplitatea dezvoltării.** Uşurinţa scrierii aplicaţiilor determinată de interfaţa API şi de limbajul aferent acesteia.
- **Portabilitate.** Dacă aplicaţia se va folosi sau nu pentru sisteme de baze de date altele decât MySQL.

Fiecare din aceşti factori va fi examinat pe larg în expunerea următoare. Reţineţi că unii factori interacţionează. De exemplu, puteţi dori o aplicaţie care se desfăşoară bine, dar la fel de importantă poate fi utilizarea unui limbaj care vă permite să dezvoltaţi rapid aplicaţia, chiar dacă nu se derulează atât de eficient.

Mediul de execuţie

Când scrieţi o aplicaţie, în general aveţi o oarecare idee cu privire la mediul în care se va folosi aceasta. De exemplu, poate fi un program de generare a rapoartelor pe care îl invocaţi din interpretor, respectiv un program de sumar al contabilităţii furnizorilor, care rulează ca sarcină a utilitarului cron la sfârşitul fiecărei luni. În general, comenzile rulate din interpretor sau din utilitarul cron sunt de sine stătătoare şi nu pun prea multe probleme mediului de execuţie. Pe de altă parte, puteţi scrie o aplicaţie care urmează a fi invocată de un server Web. Un asemenea program poate fi capabil de a extrage tipuri de informaţii foarte exacte din mediul său de dezvoltare, de genul: Ce browser foloseşte clientul? Care au fost parametrii introduşi într-un formular de solicitare a abonării la o listă de corespondenţă? A furnizat clientul parola corectă pentru accesul la informaţiile legate de personalul nostru?

Fiecare limbaj API este mai mult sau mai puţin adecvat pentru scrierea de aplicaţii în aceste medii diferite:

- C este un limbaj de uz general, deci, în principiu, poate fi folosit la orice. În practică, tendinţa este de a se utiliza limbajul C pentru programe autonome, nu pentru programarea în Web. Un motiv, probabil, este acela că în C prelucrarea textelor şi gestiunea memoriei sunt mai dificile decât în Perl sau PHP, limbaje folosite pe scară mare în aplicaţiile Web.
- Perl, ca şi C, este adecvat pentru scrierea de programe autonome. Totuşi, se întâmplă ca Perl să fie deosebit de util şi pentru dezvoltarea siturilor Web, de exemplu prin utilizarea modulului CGI.pm. Aceasta face din Perl un limbaj util pentru scrierea aplicaţiilor care stabilesc o legătură între MySQL şi Web. O asemenea aplicaţie poate interfaţa cu Web-ul prin intermediul modulului CGI.pm şi poate interacţiona cu MySQL cu ajutorul modulului DBI.
- PHP este destinat, din proiectare, pentru scrierea aplicaţiilor Web, deci reprezintă, în mod evident, limbajul cel mai propice pentru Web. Mai departe, accesul la bazele de date este una dintre cele mai puternice caracteristici ale limbajului PHP, deci acesta constituie opţiunea naturală pentru aplicaţiile Web care execută operaţii legate de MySQL. Este posibilă utilizarea limbajului PHP ca interpretor autonom (de exemplu pentru execuţia scripturilor din shell[3]), dar nu este folosit foarte frecvent în acest mod.

Date fiind aceste consideraţii, C şi Perl sunt cele mai eligibile limbaje dacă doriţi să scrieţi o aplicaţie autonomă. Pentru aplicaţiile Web, Perl şi PHP sunt cele mai adecvate. Dacă trebuie să scrieţi aplicaţii de ambele tipuri, dar nu cunoaşteţi nici unul din limbajele respective şi doriţi să învăţaţi cât mai puţin posibil, Perl poate fi alegerea dumneavoastră cea mai bună.

[3] Am folosit aici termenul *shell* în detrimentul traducerii sale uzuale (*interpretor*) pentru a nu crea o confuzie cu interpretorul PHP despre care se discută în paragraful respectiv. – N.T.

Performanţă

Dacă sunt identice din toate celelalte puncte de vedere, în general preferăm ca aplicaţiile noastre să ruleze cât mai rapid posibil. Totuşi, importanţa efectivă a performanţei tinde să fie corelată cu frecvenţa de utilizare a unui program. Pentru un program pe care îl rulaţi o dată pe lună, în timpul nopţii, sub forma unei sarcini a utilitarului `cron`, performanţa nu este un factor chiar atât de important. Dacă rulaţi un program de mai multe ori pe secundă într-un sit Web puternic solicitat, fiecare fărâmă de ineficienţă pe care o eliminaţi poate determina o diferenţă semnificativă. În cazul din urmă, performanţa joacă un rol important în utilitatea şi atractivitatea sitului dumneavoastră. Un sit lent este agasant pentru utilizatori, indiferent de conţinutul său, iar dacă situl reprezintă o sursă de venit pentru dumneavoastră, o scădere a performanţelor se converteşte direct într-o reducere a veniturilor. Nu veţi putea onora un număr atât de mare de conexiuni simultan, iar vizitatorii dezgustaţi pur şi simplu vor renunţa şi vor pleca în altă parte.

Evaluarea performanţelor este o problemă complexă. Cel mai bun indicator al comportării aplicaţiei dumneavoastră atunci când este scrisă pentru o anumită interfaţă API este de a o scrie sub acea interfaţă şi de a o folosi ca atare. Iar cel mai bun test comparativ este de a implementa aplicaţia de mai multe ori, sub interfeţe API diferite, pentru a examina diferenţele între versiuni. Desigur, nu aceasta este situaţia în realitate. Cel mai adesea, doriţi să vă vedeţi scris programul. Odată ce acesta funcţionează, vă puteţi gândi la ajustări dacă este necesar să ruleze mai repede, să folosească o cantitate mai redusă de memorie sau dacă prezintă un alt aspect care trebuie îmbunătăţit într-un alt mod. Dar există cel puţin doi factori care influenţează performanţele într-un mod relativ consecvent:

- Programele compilate se execută mai rapid decât scripturile interpretate.
- Pentru limbajele interpretate folosite într-un context Web, performanţele sunt mai bune atunci când interpretorul este invocat ca un modul care face parte din serverul Web însuşi, decât ca proces separat.

Limbaje compilate sau limbaje interpretate

Ca principiu general, aplicaţiile compilate sunt mai eficiente, folosesc o cantitate mai redusă de memorie şi se execută mai rapid decât o versiune echivalentă a programului scrisă într-un limbaj de scripting. Acest lucru se datorează suprasarcinii determinate de interpretorul limbajului care execută scripturile. Deoarece C este compilat, iar Perl şi PHP sunt interpretate, în general programele scrise în C vor rula mai rapid decât scripturile Perl sau PHP.

Pentru un program puternic solicitat, C este deseori cea mai bună opţiune. Clientul în linie de comandă `mysql` inclus în distribuţia MySQL este un bun exemplu în acest sens.

Există, desigur, şi factori care au tendinţa de a atenua această deosebire clară. Unul din aceştia este că, în general, programele scrise în C sunt mai rapide, dar este foarte posibil să scrieţi în C programe ineficiente. Scrierea unui program într-un limbaj compilat nu este un paşaport automat spre un nivel mai ridicat de performanţă. Nu sunteţi scutit de a *gândi* ceea ce aveţi de făcut. În plus, diferenţa dintre programele compilate şi cele interpretate este atenuată dacă o aplicaţie cu scripturi îşi petrece majoritatea timpului executând liniile de program din rutinele din biblioteca client MySQL care sunt legate la motorul interpretorului.

Versiuni autonome sau modulare ale interpretoarelor limbajelor

Pentru aplicaţiile bazate pe Web, interpretoarele limbajelor de scripting sunt folosite, de obicei, într-una din două forme (cel puţin în cazul Apache, serverul Web pe care îl vom folosi pentru a scrie aplicaţiile Web):

- Puteţi determina serverul Apache să invoce interpretorul ca proces separat. Când Apache trebuie să ruleze un script Perl sau PHP, porneşte programul corespunzător şi îi cere acestuia să execute scriptul. În acest caz, Apache foloseşte interpretoarele ca programe CGI; cu alte cuvinte, execută comunicarea cu programele respective folosind protocolul Common Gateway Interface (CGI).
- Interpretorul poate fi folosit ca modul care este legat direct la binarul Apache şi rulează ca parte a procesului Apache însuşi. În ceea ce priveşte serverul Apache, interpretoarele Perl şi PHP iau forma modulelor `mod_perl`, respectiv `mod_php3`.

Partizanii limbajelor Perl şi PHP vor evidenţia avantajele legate de viteză ale interpretorului lor preferat, dar toţi sunt de acord că forma în care rulează interpretorul este un factor cu mult mai important decât limbajele în sine. Ambele interpretoare rulează mult mai rapid ca modul decât ca aplicaţie CGI autonomă.

Cu o aplicaţie autonomă, este necesar să porniţi interpretorul de fiecare dată când urmează a fi executat un script, deci creaţi o penalizare semnificativă privind suprasarcina de creare de proces. Când este folosit ca modul în cadrul unui proces Apache care rulează deja, funcţionalităţile unui interpretor pot fi accesibile instantaneu din paginile dumneavoastră de Web. Acest fapt determină o creştere substanţială a performanţelor prin reducerea suprasarcinii şi se converteşte direct într-o creştere a capacităţii de tratare a cererilor recepţionate şi de distribuire rapidă a acestora.

Penalizarea la pornire a interpretorului autonom are ca rezultat, în mod caracteristic, o performanţă cu minimum un ordin de mărime mai redus decât cea a interpretorului sub formă de modul. Costul necesar la pornirea interpretorului este important mai ales dacă aveţi în vedere faptul că servirea paginii Web implică, de regulă, tranzacţii rapide cu un volum redus de prelucrări, nu tranzacţii substanţiale, cu un volum mare de prelucrări. Dacă pierdeţi mult timp cu pornirea şi mai puţin cu execuţia scriptului, atunci majoritatea resurselor dumneavoastră se irosesc. Este ca şi cum aţi pierde majoritatea zilei pregătindu-vă pentru lucru, ajungeţi la birou la ora 16 şi apoi plecaţi acasă la 17.

Vă puteţi pune întrebarea de ce apar economii de timp la versiunile modul ale interpretoarelor, deoarece tot trebuie să porniţi serverul Apache. Motivul este acela că, atunci când Apache porneşte, creează imediat un rezervor de procese copil care vor fi folosite pentru rezolvarea cererilor recepţionate. Când soseşte o cerere care implică execuţia unui script, există deja un proces Apache gata pregătit pentru a rezolva cererea. De asemenea, fiecare instanţă de Apache deserveşte mai multe cereri, deci costul procesului de pornire se calculează în raport cu un set de cereri, nu în raport cu o singură cerere.

Când Perl și PHP sunt instalate în formă modulară (sub forma `mod_perl` și `mod_php3`), care din ele se descurcă mai bine? Acest aspect constituie un subiect de controverse, dar în general se aplică următoarele linii directoare:

- Perl convertește scriptul într-o formă compilată intern; PHP nu realizează această conversie. Astfel, odată scriptul analizat, Perl are tendința de a-l executa oarecum mai rapid, mai ales pentru ciclurile cu un mare număr de iterații.
- `mod_perl` poate memora scripturile în cache, pentru a mări performanța scripturilor care se execută în mod repetat. Dacă un script se află în cache, Perl începe să-l execute mai rapid, deoarece nu mai trebuie analizat din nou. În caz contrar, PHP începe să execute mai rapid scriptul.
- `mod_perl` are o amprentă de memorie mai mare decât PHP; procesele Apache sunt mai mari dacă `mod_perl`, și nu `mod_php3`, este legat la server. PHP a fost proiectat sub considerentul că trebuie să coopereze cu alt proces și să poată fi activat și dezactivat de mai multe ori pe durata de viață a procesului respectiv. Perl a fost proiectat pentru a rula din linia de comandă, ca program autonom, nu ca limbaj destinat a fi încorporat într-un proces de server Web. Această caracteristică este cea care contribuie, probabil, la creșterea dimensiunii amprentei de memorie; ca modul, Perl pur și simplu nu rulează în mediul său natural. Alți factori care contribuie la creșterea dimensiunii amprentei de memorie sunt memorarea în cache a scripturilor și modulele Perl suplimentare pe care le folosesc scripturile. În ambele situații, în memorie este încărcată o cantitate mai mare de program, care rămâne acolo pe toată durata vieții procesului Apache.

Indiferent de avantajele pe care Perl le poate avea în raport cu PHP în ceea ce privește viteza de execuție a scripturilor, acestea sunt eliminate de PHP 4. Acesta este asemănător cu PHP 3 în ceea ce privește funcționalitățile și interfața, dar incorporează Zend, un motor de interpretor cu performanțe superioare.

În orice caz, toți acești factori au ca rezultat diferențe de ordinul zecimilor de procent între performanțele versiunilor de tip modul ale limbajelor Perl și PHP. Cel mai important este să evitați interpretoarele autonome ori de câte ori este posibil, indiferent de limbajul pe care îl alegeți.

Versiunea autonomă a unui interpretor are un avantaj în raport cu omologul său în versiune modul, în sensul că puteți determina rularea primei versiuni sub un alt identificator de utilizator. Versiunile modul rulează întotdeauna sub același identificator de utilizator ca și serverul Web, care, în mod caracteristic, este un cont cu privilegii minimale, din motive de securitate (de exemplu dacă trebuie să puteți citi sau scrie în fișiere protejate). Puteți combina abordările autonomă și cea de tip modul, dacă doriți, folosind versiunea modul în mod prestabilit, respectiv versiunea autonomă pentru situații în care este necesar ca scripturile să ruleze cu privilegiile unui anumit utilizator.

Reducerea cerințelor de memorie ale modulului `mod_perl`

Există tehnici care vă permit să activați numai anumite procese Apache pentru `mod_perl`. Astfel, suprasarcina determinată de plusul de memorie apare numai la acele procese care execută scripturi Perl. Domeniul dedicat modulului `mod_perl` din situl Web Apache conține o expunere pertinentă a diferitelor strategii pe care le puteți alege. (Pentru mai multe informații, vezi `http://perl.apache.org/guide/`.)

Aceasta înseamnă cu atât mai mult că, ori de câte ori folosiți Perl sau PHP, trebuie să încercați să folosiți limbajul respectiv dintr-un modul Apache, nu să invocați un proces de interpretor separat. Rezervați utilizarea interpretorului autonom numai pentru acele cazuri unde nu se poate folosi versiunea modul, cum sunt scripturile care necesită privilegii speciale. Pentru aceste situații, vă puteți prelucra scriptul folosind mecanismul Apache suEXEC, pentru a porni interpretorul sub un anumit identificator de utilizator.

Timp de dezvoltare

Factorii pe care i-am descris anterior afectează performanța aplicațiilor dumneavoastră, dar este posibil ca eficiența brută a execuției să nu fie unicul dumneavoastră scop. Timpul este de asemenea important, ca și simplitatea în programare, deci un alt factor care trebuie luat în considerare la selectarea unei interfețe API pentru programarea în MySQL este rapiditatea cu care vă puteți desfășura aplicațiile. Dacă puteți scrie un script Perl într-un timp echivalent cu jumătatea intervalului necesar pentru a dezvolta echivalentul în C al programului respectiv, atunci puteți prefera să folosiți interfața API pentru modulul Perl DBI decât interfața API pentru C, chiar dacă aplicația rezultantă nu rulează la fel de rapid. Deseori, este normal să fiți mai puțin interesat de timpul de execuție a programului decât de timpul pe care îl alocați scrierii acestuia, mai ales pentru aplicațiile care nu sunt executate frecvent. O oră din timpul dumneavoastră este cu mult mai valoroasă decât o oră din timpul calculatorului[4]!

În general, limbajele de scripting vă permit să „puneți pe roate“ mult mai rapid un anumit program, mai ales pentru elaborarea unui prototip al aplicației finite. Cel puțin doi factori contribuie la aceasta.

Mai întâi, limbajele de scripting au tendința de a furniza construcții de nivel mai înalt. Acest lucru vă permite să gândiți la un nivel de abstractizare mai ridicat, astfel încât să vă puteți concentra asupra a ceea ce aveți de făcut, nu asupra detaliilor necesare. De exemplu, tablourile asociative (hash-urile) din Perl determină o economie de timp semnificativă pentru întreținerea datelor cu o relație de tip cheie-valoare (cum sunt perechile identificator elev – nume elev). Limbajul C nu dispune de o asemenea construcție. Dacă ați fi dorit să implementați o asemenea entitate în C, ar fi trebuit să scrieți programe pentru rezolvarea mai multor detalii de nivel scăzut, legate de aspecte precum gestiunea memoriei și manipularea șirurilor, programe pe care trebuie să le depanați. Ori, așa ceva ia timp.

În al doilea rând, ciclul de dezvoltare are mai puține etape pentru limbajele de scripting. În C, atunci când vă dezvoltați aplicațiile, trebuie să parcurgeți testul obișnuit de editare-compilare-testare. De fiecare dată când modificați programul, trebuie să-l compilați din nou înainte de a-l testa. Cu Perl și PHP, ciclul de dezvoltare se rezumă la editare și testare, deoarece puteți rula un script imediat după fiecare modificare, fără compilare. Pe de altă parte, compilatorul impune mai multe restricții asupra programului dumneavoastră, sub forma unei verificări mai stricte a tipurilor. Necesitatea disciplinei mai stricte impuse de compilator vă poate ajuta să evitați hibe care nu sunt atât de ușor de depistat în limbaje mai liberale, precum Perl sau PHP. Dacă scrieți greșit un nume de variabilă în C, compi-

[4] Ce ne facem, însă, dacă acea oră din timpul calculatorului este o oră din timpul clientului? – N.T.

latorul vă va avertiza. PHP nu va face acest lucru, iar Perl nu vă va avertiza decât dacă îi cereți aceasta. Aceste restricții mai puternice sunt valoroase mai ales dacă aplicațiile dumneavoastră devin mai mari și mai dificil de întreținut.

În general, compromisul este cel uzual dintre limbajele compilate și cele interpretate, între timpul de dezvoltare și performanță: Doriți să dezvoltați programul folosind un limbaj compilat, astfel încât să se execute mai rapid atunci când rulează, dar să pierdeți mai mult timp la scrierea programului? Sau doriți să scrieți programul sub formă de script, pentru a-l putea exploata în cel mai scurt timp, chiar și cu prețul unei scăderi a vitezei de execuție?

De asemenea, este posibilă combinarea celor două metode. Scrieți un script drept „ciornă" pentru a dezvolta rapid un prototip al aplicației, în vederea testării logicii și pentru a vă asigura că algoritmii sunt corecți. Dacă programul se dovedește a fi util și este executat suficient de frecvent astfel încât performanța să devină o problemă, îl puteți rescrie sub formă de aplicație compilată. Astfel, puteți obține avantaje din ambele părți: o creare rapidă a prototipului pentru dezvoltarea inițială a aplicației și un nivel de performanță optim pentru produsul final.

Într-un sens mai strict, nici una din interfețele API pentru Perl DBI și PHP nu oferă caracteristici care nu există în biblioteca client C. Aceasta deoarece ambele interfețe API obțin acces la MySQL prin legarea bibliotecii C pentru MySQL la interpretoarele Perl și PHP. Totuși, mediul în care sunt înglobate caracteristicile sistemului MySQL este foarte diferit pentru C decât pentru Perl sau PHP. Gândiți-vă la unele dintre operațiile necesare la interacțiunea cu serverul MySQL și întrebați-vă în ce măsură fiecare dintre limbajele API vă poate ajuta la efectuarea acestora. Iată câteva exemple:

- **Gestiunea memoriei.** În C, veți lucra cu `malloc()` și `free()` pentru toate sarcinile care implică structuri de date alocate dinamic. Perl și PHP execută aceste operații automat. De exemplu, dimensiunile tablourilor cresc automat, iar șirurile cu lungime dinamică se pot folosi fără a vă mai gândi la gestiunea memoriei.
- **Manipularea textelor.** Perl are cele mai evoluate caracteristici în acest domeniu, iar PHP îl secondează îndeaproape. În comparație cu cele două limbaje, C este foarte rudimentar.

Desigur, în C vă puteți scrie propriile dumneavoastră biblioteci, care să încapsuleze sarcini precum gestiunea memoriei și prelucrarea textelor în funcții care să faciliteze îndeplinirea sarcinii. Va trebui însă să depanați acele funcții, și doriți ca algoritmii dumneavoastră să fie, de asemenea, eficienți. Din ambele puncte de vedere, se poate presupune că algoritmii din Perl și PHP pentru aceste operații sunt în general bine depanați și de o eficiență rezonabilă, din moment ce au avut „privilegiul" de a fi fost examinați de numeroase perechi de ochi. Puteți economisi timp beneficiind de timpul pe care alții l-au investit în această sarcină. (Pe de altă parte, dacă se întâmplă ca interpretorul să aibă o hibă, s-ar putea să trebuiască să coabitați cu aceasta până când problema este remediată. Când scrieți programe în C, aveți un control mai fin asupra comportării programului dumneavoastră.)

Limbajele diferă din punct de vedere al „siguranței". Interfața API pentru C furnizează interfața de nivel cel mai scăzut cu serverul și aplică politica cea mai puțin pretențioasă. Din acest motiv, furnizează siguranța cea mai precară. Dacă executați funcții API defecte,

puteți avea noroc și primiți o eroare „de nesincronizare" sau puteți avea ghinion și programul dumneavoastră va cădea. Scriptul dumneavoastră cade dacă nu efectuați operațiile în ordinea corectă, dar interpretorul nu se defectează. O altă sursă fertilă de hibe din programele C care determină căderea sistemului este utilizarea memoriei alocate dinamic și a pointerilor asociați acesteia. Perl și PHP se ocupă automat de gestiunea memoriei, deci este mult mai puțin probabil ca scripturile dumneavoastră „să-și dea duhul" datorită unor hibe de gestiune a memoriei.

Timpul de dezvoltare este afectat de volumul de suport extern disponibil pentru un limbaj. Suportul extern pentru C există sub forma unor biblioteci container care încapsulează funcții API în C pentru MySQL în rutine care sunt mai ușor de folosit. Bibliotecile care execută această operație sunt disponibile atât pentru C, cât și pentru C++. Indiscutabil, Perl are cea mai mare cantitate de module add-on, sub formă de module Perl (conceptual similare cu modulele Apache). Există chiar o infrastructură, concepută pentru a facilita localizarea și obținerea acestor module (CPAN sau Comprehensive Perl Archive Network). Folosind module Perl, obțineți acces la toate categoriile de funcții fără a scrie nici măcar o linie de program. Doriți să scrieți un script care să genereze un raport dintr-o bază de date, care apoi să fie trimis cuiva sub formă de fișier atașat la un e-mail? Procurați-vă unul dintre modulele MIME și veți dispune de o funcționalitate instantanee de generare a fișierelor atașate.

PHP nu dispune de aceeași cantitate de suport extern (ceea ce nu este surprinzător, dat fiind că este un limbaj mai recent). Poate că modulul add-on cel mai bine cunoscut este PHP Base Library (PHPLIB). Să presupunem că scrieți o aplicație Web care trebuie să limiteze accesul la anumite pagini Web numai pentru utilizatorii autorizați, în funcție de un anumit mecanism bazat pe nume și parolă. Puteți scrie programul de suport pentru aplicația respectivă în orice limbaj, dar, dacă folosiți PHP Base Library (PHPLIB), nu va trebui să vă pierdeți vremea redescoperind roata. PHPLIB furnizează metode de autentificare și de asemenea vă permite să urmăriți utilizatorii neautorizați pe parcursul unei sesiuni (o succesiune de deschideri de pagină de la un client dat, tratate ca părți ale unei singure vizite logice). De asemenea, puteți atribui permisiuni utilizatorilor, ceea ce vă permite să efectuați operații precum definirea unor utilizatori administrativi, care dispun de mai multe privilegii.

Portabilitate

Problema portabilității este legată de ușurința cu care un program scris pentru a obține accesul la motorul MySQL poate fi modificat astfel încât să folosească un alt motor. Acesta poate fi un aspect căruia nu-i acordați nici o importanță. Totuși, cu excepția cazului în care aveți calități de vizionar și afirmați „Niciodată nu voi folosi acest program cu un alt sistem de baze de date decât MySQL", poate fi riscant: să presupunem că vă schimbați serviciul și doriți să vă folosiți vechile programe, dar dacă noul dumneavoastră patron folosește un alt sistem de baze de date?

Dacă portabilitatea este o prioritate, trebuie să aveți în vedere diferențele nete între interfețele API:

- DBI asigură interfața API cea mai portabilă, deoarece independența de sistemul de baze de date este un deziderat explicit de proiectare pentru DBI.
- PHP este mai puțin portabil, deoarece nu furnizează același tip de interfață uniformă pentru diferitele sisteme de baze de date ca și DBI. Apelurile de funcții PHP pentru fiecare sistem de baze de date acceptat sunt asemănătoare cu apelurile corespunzătoare din interfața de bază API pentru C. Există o oarecare atenuare a diferențelor, dar, cel puțin, va trebui să modificați numele funcțiilor de manipulare a bazelor de date pe care le invocați. De asemenea, s-ar putea să fiți obligat să revizuiți puțin și logica aplicației dumneavoastră, deoarece interfețele cu diferitele sisteme de baze de date nu funcționează toate în exact același mod.
- Interfața API pentru C asigură portabilitatea cea mai redusă între sistemele de baze de date. Prin natura sa, această interfață este proiectată și destinată special utilizării cu MySQL.

Portabilitatea – sub forma independenței de sistemul de baze de date – este deosebit de importantă când trebuie să obțineți acces la mai multe sisteme de baze de date din cadrul aceleiași aplicații. Aici pot fi incluse sarcini simple, precum mutarea datelor dintr-un SGBDR în altul, respectiv operații mai complexe, cum ar fi generarea unui raport în funcție de informații obținute de la un număr de sisteme de baze de date.

Atât DBI, cât și PHP furnizează suport pentru accesul la mai multe motoare de baze de date, deci vă puteți conecta simultan, cu ușurință, la servere pentru sisteme de baze de date diferite, plasate chiar pe calculatoare diferite. Cu toate acestea, DBI și PHP nu sunt la fel de adecvate pentru sarcini care implică regăsirea și prelucrarea datelor din mai multe sisteme de baze de date complet diferite. DBI este de preferat, deoarece folosiți un singur set de apeluri de acces, indiferent de sistemele de baze de date pe care le folosiți. Să presupunem că doriți să transferați date între sistemele de baze de date MySQL, mSQL și Postgres. Cu DBI, unica diferență între modurile de utilizare a celor trei sisteme de baze de date este apelul `DBI->connect()` folosit pentru conectarea la fiecare server. Cu PHP, aveți un script mai complicat, care încorporează trei seturi de apeluri pentru citire, respectiv trei seturi de apeluri pentru scriere.

Un caz extrem al aplicației cu baze de date multiple este scriptul `crash-me` din distribuția MySQL, care testează funcționalitățile a numeroase programe server de baze de date diferite. Acest script este scris folosind DBI, care reprezintă opțiunea evidentă pentru o asemenea aplicație, deoarece puteți obține acces în același mod la toate sistemele de baze de date.

CAPITOLUL 6

Interfața API MySQL pentru C

MySQL furnizează o bibliotecă client scrisă în limbajul de programare C, pe care o puteți folosi pentru a scrie programe client care obțin accesul la bazele de date MySQL. Această bibliotecă definește o interfață de programare a aplicațiilor, care include următoarele facilități:

- Rutine de gestiune a conexiunii, pentru inițierea și terminarea unei sesiuni cu un server.
- Rutine de construcție a interogărilor, de trimitere a acestora la server și de prelucrare a rezultatelor.
- Funcții de raportare a stării și a erorilor, pentru determinarea motivului exact al erorii atunci când alte apeluri de funcții din interfața API pentru C eșuează.

Acest capitol vă prezintă modul de utilizare a bibliotecii client pentru scrierea propriilor dumneavoastră programe. Unele dintre dezideratele de care vom ține cont sunt consecvența cu programele client existente din distribuția MySQL, precum și caracterul modular și capacitatea de reutilizare a codului. Presupun că aveți cunoștințe despre programarea în C, dar am încercat să nu vă confund cu un expert în materie.

Capitolul dezvoltă o serie de programe client într-o progresie simplă, de la foarte simple la programe mai complexe. Prima parte a acestei progresii dezvoltă cadrul pentru un schelet al unui program client, care nu face decât să se conecteze și să se deconecteze de la server. Motivul este că, deși programele client MySQL sunt scrise cu scopuri diferite, toate au un lucru în comun: stabilesc o conexiune cu serverul.

Vom construi scheletul în etape:

1. Scriem un program elementar de conexiune și întrerupere a conexiunii (`client1`).
2. Adăugăm logică de verificare a apariției erorilor (`client2`).
3. Facem programul de conexiune modular și reutilizabil (`client3`).
4. Adăugăm capacitatea de a obține parametri de conexiune (gazdă, utilizator, parolă) la rulare (`client4`).

Acest cadru este suficient de general și dumneavoastră îl puteți folosi drept bază pentru orice număr de programe client. După ce-l vom crea, vom face o pauză pentru a ne gândi la modul de manipulare a diverselor interogări. La început, vom discuta despre modul de manipulare a anumitor instrucțiuni SQL codate hard (adică înglobate în codul programului – N.T.), după care vom crea un program care se poate folosi pentru prelucrarea instrucțiunilor arbitrare. Apoi, vom adăuga programul de prelucrare a interogărilor la cadrul nostru de program client, pentru a crea un alt program (`client5`) care este similar clientului `mysql`.

De asemenea, vom lua în considerare (și vom rezolva) unele probleme comune, de genul „Cum îmi pot procura informații despre structura tabelelor mele?" sau „Cum pot insera imagini în baza mea de date?".

Capitolul de față discută despre funcții și tipuri de date din biblioteca client numai în măsura necesităților. Pentru o listă comprehensivă a tuturor funcțiilor și tipurilor, vezi Anexa F, „Referință API C". Puteți folosi anexa respectivă ca referință pentru alte cunoștințe privind orice componentă a bibliotecii client pe care încercați să o folosiți.

Programele date ca exemplu pot fi descărcate din Internet, deci le puteți încerca direct, fără a fi necesar să le tastați. Pentru instrucțiuni, vezi Anexa A, „Obținerea și instalarea programelor".

Unde se găsesc exemplele

O întrebare comună care se află în lista de corespondență MySQL este: „Unde pot găsi unele exemple de programe client scrise în C?" Desigur, răspunsul este: „Chiar aici, în cartea asta". Dar un fapt pe care mulți au tendința să nu-l ia în considerare este acela că distribuția MySQL conține numeroase programe client (`mysql`, `mysqladmin` și `mysqldump`, de exemplu), din care majoritatea sunt scrise în C. Deoarece distribuția este disponibilă sub formă de cod sursă, MySQL însuși vă oferă o cantitate semnificativă de exemple de programe client. Ca atare, dacă nu ați făcut-o până acum, luați o distribuție în cod sursă și examinați programele din catalogul `client`. Programele client MySQL aparțin domeniului public și puteți împrumuta liniștit linii de cod din ele pentru propriile dumneavoastră programe.

Între exemplele furnizate în acest capitol și programele client incluse în distribuția MySQL, puteți găsi ceva similar cu intențiile dumneavoastră atunci când vă scrieți propriile programe. În acest caz, puteți refolosi linii de program prin copierea unui program existent și modificarea sa. Trebuie să citiți acest capitol pentru a înțelege modul de funcționare a bibliotecii client. Rețineți, totuși, că nu este necesar să scrieți întotdeauna toate programele, pornind de la zero. (Veți observa că posibilitatea de reutilizare a codului este unul dintre scopurile urmărite în expunerea noastră din acest capitol privind scrierea programelor.) Dacă puteți economisi o mare cantitate de muncă pornind de la realizările unei alte persoane, cu atât mai bine.

Procedura generală pentru construirea programelor client

Această secțiune descrie etapele necesare pentru compilarea și legarea unui program care folosește biblioteca client MySQL. Comenzile pentru construirea clienților sunt oarecum diferite de la un sistem la altul, iar dumneavoastră va trebui să modificați puțin comenzile prezentate aici. Totuși, descrierea are un caracter general și trebuie să o puteți aplica aproape oricărui program client pe care îl scrieți.

Cerințe elementare de sistem

Când scrieți un program client MySQL în C, veți avea nevoie de un compilator de C, evident. Exemplele prezentate aici folosesc compilatorul `gcc`. De asemenea, veți avea nevoie de următoarele entități în afară de propriile dumneavoastră fișiere sursă:

- Fișierele antet MySQL
- Biblioteca client MySQL

Fișierele antet și biblioteca client MySQL constituie suportul pentru programarea clienților. Acestea pot fi deja instalate în sistemul dumneavoastră. În caz contrar, este nevoie să vi le procurați. Dacă MySQL a fost instalat dintr-o distribuție sursă sau binară, suportul pentru programarea clienților trebuie să fi fost deja instalat ca parte a procesului de instalare a distribuției. Dacă MySQL a fost instalat din fișiere RPM, acest suport nu va exista decât cu condiția ca fișierele RPM pentru dezvoltatori să fi fost deja instalate. Dacă trebuie să instalați fișierele antet și biblioteca MySQL, vezi Anexa A.

Compilarea și legarea programului client

Pentru a compila și a lega un program client, trebuie să specificați unde se găsesc fișierele antet și biblioteca client MySQL, deoarece aceste componente, de obicei, nu sunt instalate în locațiile unde compilatorul și programul de legare le caută în mod prestabilit. Pentru exemplul următor, să presupunem că locațiile fișierelor antet și a bibliotecii client sunt `usr/local/include/mysql`, respectiv `/usr/local/lib/mysql`.

Pentru a arăta compilatorului cum să găsească fișierele antet MySQL, transmiteți-i un argument `-I/usr/local/include/mysql` atunci când compilați un fișier sursă într-un fișier obiect. De exemplu, puteți folosi o comandă ca aceasta:

```
% gcc -c -I/usr/local/include/mysql myclient.c
```

Pentru a indica programului de legare unde să găsească biblioteca client și care este numele acesteia, transmiteți argumentele `-L/usr/local/lib/mysql` și `-lmysqlclient` atunci când legați fișierul obiect, pentru a produce un fișier binar executabil, după cum urmează:

```
% gcc -o myclient myclient.o -L/usr/local/lib/mysql -lmysqlclient
```

În cazul în care clientul dumneavoastră este alcătuit din mai multe fișiere, denumiți toate fișierele obiect în comanda de legare. Dacă etapa de legare generează o eroare legată de imposibilitatea de a găsi funcția `floor()`, legați biblioteca de funcții matematice prin adăugarea opțiunii `-lm` la sfârșitul comenzii:

```
% gcc -o myclient myclient.o -L/usr/local/lib/mysql -lmysqlclient -lm
```

Mai poate fi necesară și adăugarea altor biblioteci. De exemplu, poate veți avea nevoie de `-lsocket -lnsl` pe sistemele Solaris.

Dacă nu folosiți `make` pentru construirea programelor, vă propun să învățați să folosiți această comandă, astfel încât să nu fiți obligat să tastați manual o mulțime de comenzi pentru construirea programelor. Să presupunem că aveți un program client, denumit `myclient`, care constă din două fișiere sursă, `main.c` și `aux.c`, și un fișier antet, denumit `myclient.h`. O comandă `Makefile` simplă pentru construirea acestui program se poate prezenta astfel:

```
CC = gcc
INCLUDES = -I/usr/local/include/mysql
LIBS = -L/usr/local/lib/mysql -lmysqlclient

all: myclient

main.o: main.c myclient.h
    $(CC) -c $(INCLUDES) main.c
aux.o: aux.c myclient.h
```

Continuare

Continuare

```
	$(CC) -c $(INCLUDES) aux.c

myclient: main.o aux.o
	$(CC) -o myclient main.o aux.o $(LIBS)

clean:
	rm -f myclient main.o aux.o
```

Dacă în sistemul dumneavoastră este necesară legarea bibliotecii de formule matematice, modificați valoarea parametrului `LIBS`, adăugând `-lm` la sfârșit:

```
LIBS = -L/usr/local/lib/mysql -lmysqlclient -lm
```

Dacă aveți nevoie și de alte biblioteci, cum sunt `-lsocket` sau `-lnsl`, adăugați-le și pe acestea la opțiunea `LIBS`.

Folosind `Makefile`, vă puteți reconstrui programul ori de câte ori modificați vreuna dintre fișierele sursă, prin simpla tastare a expresiei "`make`". Este un procedeu mai simplu și mai puțin supus la erori decât dacă tastați o comandă gcc lungă.

Client 1 – Conectarea la server

Primul nostru program client MySQL este cât se poate de simplu: se conectează la un server, se deconectează și își încheie execuția. Acest program nu este foarte util în sine, dar trebuie să știți cum să efectuați operațiile respective, deoarece, înainte de a putea exploata efectiv în orice mod o bază de date MySQL, trebuie să fiți conectat la un server. Aceasta este o operație atât de comună, încât veți folosi programul pe care îl creați pentru a stabili o conexiune la fiecare client pe care îl veți scrie. De asemenea, această sarcină ne permite să începem de la ceva simplu. Putem extinde ulterior programul client, pentru a executa operații utile.

Sursa pentru primul nostru program client, `client1`, este compusă dintr-un singur fișier, `client1.c`:

```
/* client1.c */

#include <stdio.h>
#include <mysql.h>

#define def_host_name NULL     /* gazda la care se va stabili conexiunea
                                  (valoare prestabilita = localhost) */
#define def_user_name NULL     /* nume utilizator (valoare
                                  prestabilita = numele dumneavoastra
                                  de deschidere a sesiunii de lucru) */
#define def_password NULL      /* parola (valoare prestabilita = nici
                                  una) */
#define def_db_name NULL       /* baza de date de utilizat (valoare
                                  prestabilita = nici una) */

MYSQL *conn;                    /* pointer spre variabila de
                                   tratare a conexiunii */

int
main (int argc, char *argv[])
{
   conn = mysql_init (NULL);
   mysql_real_connect (
          conn,               /* pointer spre variabila de
                                 tratare a conexiunii */
          def_host_name,      /* gazda la care se va stabili
                                 conexiunea */
          def_user_name,      /* numele utilizatorului */
          def_password,       /* parola */
          def_db_name,        /* baza de date de utilizat */
          0,                  /* port (se va folosi valoarea
                                 prestabilita) */
          NULL,               /* soclu (se va folosi
                                 valoarea prestabilita) */
          0);                 /* indicatoare (nici unul) */
   mysql_close (conn);
   exit(0);
}
```


Fișierul sursă începe prin a include `stdio.h` și `mysql.h`. Clienții MySQL pot include și alte fișiere antet, dar, în general, acestea două constituie minimul necesar.

Valorile prestabilite pentru numele gazdei, numele utilizatorului, parola și numele bazei de date sunt codate hard în program, pentru simplitate. Ulterior, vom parametriza aceste valori, astfel încât dumneavoastră să le puteți specifica în fișiere cu opțiuni sau în linia de comandă.

Funcția `main()` a programului inițiază și încheie conexiunea cu serverul. Stabilirea unei conexiuni este un proces compus din două etape:

1. Apelarea funcției `mysql_init()` pentru a obține o variabilă de tratare a conexiunii. Tipul de date `MYSQL` este o structură care conține informații despre o conexiune. Variabilele de acest tip se numesc variabile de tratare a conexiunii. Când transferați o valoare `NULL` funcției `mysql_init()`, aceasta alocă o variabilă `MYSQL`, o inițializează și returnează un pointer la aceasta.
2. Apelarea funcției `mysql_real_connect()` pentru a stabili o conexiune cu serverul. Funcția `mysql_real_connect()` ia o mulțime de parametri:
   - Un pointer spre variabila de tratare a conexiunii. Acesta trebuie să fie `NULL`; trebuie să fie valoarea returnată de `mysql_init()`;
   - Gazda serverului. Dacă specificați `NULL` sau gazda "`localhost`", clientul se conectează la serverul care rulează pe gazda locală, folosind un soclu UNIX. Dacă specificați un nume al gazdei sau o adresă IP a gazdei, clientul se va conecta la gazda specificată folosind o conexiune TCP/IP.

     În Windows, comportarea este similară, cu deosebirea că în locul soclurilor UNIX se folosesc conexiuni TCP/IP. (În Windows NT, înainte de TCP/IP se încearcă o conexiune folosind un canal denumit, dacă gazda este `NULL`.)

- Numele de utilizator și parola. Dacă numele este `NULL`, biblioteca client trimite serverului numele dumneavoastră de deschidere a sesiunii de lucru. Dacă parola este `NULL`, nu se va trimite nici o parolă.
- Numărul portului și fișierul soclu. Acestea sunt specificate ca fiind `0`, respectiv `NULL`, pentru a indica bibliotecii client să-și folosească valorile prestabilite. Dacă portul și soclul nu sunt specificate, valorile prestabilite sunt determinate în conformitate cu gazda la care doriți să vă conectați. Detaliile acestei operații sunt date în descrierea funcției `mysql_real_connect()` din Anexa F.
- Valoarea indicatorilor (flags). Aceasta este `0`, deoarece nu folosim nici un fel de opțiuni de conexiune speciale. Opțiunile care sunt disponibile pentru acest parametru sunt discutate mai detaliat în informațiile referitoare la funcția `mysql_real_connect()` din Anexa F.

Pentru a termina conexiunea, transmiteți funcției `mysql_close()` un pointer la variabila de tratare a conexiunii. O variabilă de tratare a conexiunii care este alocată automat de către `mysql_init()` este dealocată automat atunci când transferați variabila funcției `mysql_close()` pentru încheierea conexiunii.

Pentru a încerca programul `client1`, compilați-l și legați-l folosind instrucțiunile prezentate anterior în capitolul de față pentru construcția programelor client, după care rulați-l:

```
% client1
```

Programul se conectează la server, se deconectează și își încheie execuția. Nu-i foarte interesant, dar este un început. Totuși, nu este *decât* un început, deoarece există două neajunsuri semnificative:

- Clientul nu execută nici un fel de verificare a apariției erorilor, deci nu știți dacă funcționează sau nu!
- Parametrii de conexiune (nume de gazdă, nume de utilizator etc.) sunt codați hard în codul sursă. Este mai bine să permiteți utilizatorului să-i redefinească, prin specificarea parametrilor într-un fișier cu opțiuni sau în linia de comandă.

Nici una din aceste probleme nu este dificil de rezolvat. Ne vom ocupa de ambele în următoarele câteva secțiuni.

Client 2 – Adăugarea logicii de verificare a apariției erorilor

Cel de-al doilea client al nostru va fi asemănător cu primul, numai că va fi modificat pentru a lua în calcul posibilitatea apariției erorilor. Este un obicei frecvent ca în cărțile de programare să se spună „Verificarea apariției erorilor este lăsată ca exercițiu pentru cititor“, deoarece verificarea apariției erorilor este – să o recunoaștem – o mare plictiseală. Cu toate acestea, prefer să promovez punctul de vedere conform căruia programele client MySQL trebuie să testeze condițiile de eroare și să reacționeze la acestea în mod adecvat. Apelurile la biblioteca client care returnează valori de stare procedează astfel dintr-un anumit motiv, iar dacă dumneavoastră le ignorați, o faceți pe risc propriu. Veți sfârși fie încercând să depistați probleme obscure care survin în programele dumneavoastră datorită inexistenței unei verificări a apariției erorilor, fie utilizatorii programelor dumneavoastră se vor întreba de ce programele astea au luat-o razna, fie ambele.

Să luăm în considerare programul nostru `client1`. Cum știți dacă acesta s-a conectat sau nu la server? Ați putea afla căutând în jurnalul serverului evenimentele `Connect` și `Quit`, care corespund intervalului de timp în care s-a rulat programul:

```
990516 21:52:14    20 Connect    paul@localhost on
                   20 Quit
```

Alternativ, puteți vedea în schimb un mesaj `Access denied` (acces interzis):

```
990516 22:01:47    21 Connect    Access denied for user:
                                 'paul@localhost'(Using password:NO)
```

Acest mesaj arată că nu a fost stabilită nici o conexiune. Din păcate, programul `client1` nu ne spune care dintre aceste două rezultate s-a produs. De fapt, nu poate. Nu execută nici o verificare a apariției erorilor, deci nu știe nici el ce s-a întâmplat. În orice caz, categoric nu trebuie să examinați jurnalul pentru a afla dacă ați reușit sau nu să vă conectați la server! Să remediem chiar acum problema.

Rutinele din biblioteca client MySQL care returnează o valoare indică, în general, reușita sau eșecul într-unul din următoarele două moduri:

- Funcțiile cu valori pointeri returnează un pointer diferit de `NULL` pentru succes, respectiv `NULL` pentru eșec. (În acest context, `NULL` înseamnă „Un pointer C `NULL`“ nu „O valoare `NULL` dintr-o coloană MySQL“.)

  Dintre rutinele din biblioteca client pe care le-am folosit până acum, `mysql_init()` și `mysql_real_connect()`returnează ambele un pointer al variabilei de tratare a conexiunii pentru a indica succesul, respectiv `NULL` pentru eșec.
- Funcțiile cu valori întregi returnează de obicei `0` pentru reușită și o valoare diferită de zero pentru eșec. Este important să nu se verifice existența unor anumite valori diferite de zero, cum este `-1`. Nu există nici o garanție că o funcție din biblioteca client returnează o anumită valoare atunci când eșuează. Ocazional, puteți vedea programe mai vechi, care testează incorect o valoare returnată, astfel:

```
if (mysql_XXX() == -1)          /* acest test este incorect */
    fprintf (stderr, "s-a intamplat ceva rau\n");
```

Acest test s-ar putea să funcționeze, dar s-ar putea să nu funcționeze. Interfața API MySQL nu specifică faptul că orice eroare returnată diferită de zero va fi o anumită valoare, în afara faptului că este diferită de zero. Testul trebuie scris astfel:

```
if (mysql_XXX())            /* acest test este corect */
    fprintf (stderr, "s-a intamplat ceva rau\n");
```

sau astfel:

```
if (mysql_XXX() != 0)           /* acest test este corect */
    fprintf (stderr, "s-a intamplat ceva rau\n");
```

Cele două teste sunt echivalente. Dacă examinați codul sursă al sistemului MySQL însuși, veți observa că în general se folosește prima formă a testului, care se scrie mai rapid.

Nu fiecare apel API returnează o valoare. Cealaltă rutină client pe care am folosit-o, `mysql_close()`, este una care nu returnează nici o valoare. (Cum de a putut să eșueze? Iar dacă a eșuat, ce-mi pasă mie? Oricum ați terminat conexiunea.)

Când un apel la o bibliotecă client eşuează şi aveţi nevoie de mai multe informaţii despre eşec, două apeluri de funcţii din interfaţa API sunt utile. `mysql_error()` returnează un şir care conţine un mesaj de eroare, iar `mysql_errno()` returnează un cod numeric de eroare. Trebuie să apelaţi aceste funcţii imediat după apariţia unei erori, deoarece, dacă emiteţi un alt apel API care returnează o stare, toate informaţiile de eroare pe care le veţi primi de la funcţiile `mysql_error()`sau `mysql_errno()` se vor referi la apelul ulterior.

În general, pentru utilizatorul unui program şirul de eroare este mai edificator decât codul de eroare. Dacă veţi raporta numai una din cele două informaţii, ar fi bine ca aceasta să fie şirul. Din motive de completitudine, exemplele din acest capitol raportează ambele valori.

Luând în considerare discuţia anterioară, vom scrie al doilea program client, `client2`. Acesta este similar cu `client1`, dar conţine în plus un program adecvat de verificare a apariţiei erorilor. Fişierul sursă, `client2.c`, se prezintă astfel:

```
/* client2.c */

#include <stdio.h>
#include <mysql.h>

#define def_host_name NULL     /* gazda la care se va stabili
                                  conexiunea(valoare
                                  prestabilita = localhost) */
define def_user_name NULL      /* nume utilizator (valoare
                                  prestabilita = numele dumneavoastra
                                  de deschidere a sesiunii de lucru) */
#define def_password NULL      /* parola (valoare prestabilita =
                                  nici una) */
#define def_db_name NULL       /* baza de date de utilizat (valoare
                                  prestabilita = nici una) */

MYSQL *conn;                   /* pointer spre variabila de
                                  tratare a conexiunii */


int
main (int argc, char *argv[])
{
  conn = mysql_init (NULL);
  if (conn == NULL)
  {
    fprintf(stderr, "mysql_init() ratat (probabil din lipsa
     de memorie)\n");
    exit(1);
  }
  if (mysql_real_connect (
          conn,                /* pointer spre variabila de
                                  tratare a conexiunii */
          def_host_name,       /* gazda la care se va stabili
                                  conexiunea */
          def_user_name,       /* numele utilizatorului */
          def_password,        /* parola */
          def_db_name,         /* baza de date de utilizat */
          0,                   /* port (se va folosi valoarea
                                  prestabilita) */
          NULL,                /* soclu (se va folosi
                                  valoarea prestabilita) */
          0);                  /* indicatoare (nici unul) */
      == NULL)
  {
    fprintf (stderr, "mysql_real_connect() failed: \nError %u (%s)\n",
          mysql_errno(conn), mysql_error(conn));
    exit(1);
  }
  mysql_close (conn);
  exit(0);
}
```

Logica de verificare a apariţiei erorilor se bazează pe faptul că atât `mysql_init()` cât şi `mysql_real_connect()` returnează `NULL` dacă eşuează. Reţineţi că, deşi programul verifică valoarea returnată a funcţiei `mysql_init()`, dacă aceasta eşuează nu este apelată nici o funcţie de raportare a erorilor. Aceasta deoarece nu se poate presupune că variabila de tratare a conexiunii va conţine informaţii utile atunci când `mysql_init()` eşuează. Prin contrast, atunci când `mysql_real_connect()` eşuează, variabila de tratare a conexiunii nu mai reflectă o conexiune validă, dar conţine informaţii de eroare care pot fi transmise funcţiilor de raportare a erorilor. (Nu transmiteţi variabila de tratare altor rutine client! Deoarece acestea presupun, în general, că acea conexiune este validă, programul va suferi o cădere.)

Compilaţi şi legaţi programul `client2`, după care încercaţi să-l rulaţi:

```
% client2
```

Dacă programul `client2` nu generează nici un fel de date de ieşire (aşa cum s-a arătat anterior), înseamnă că s-a conectat cu succes. Pe de altă parte, s-ar putea să vedeţi următorul rezultat:

```
% client2
mysql_real_connect() failed:
Error 1045 (Access denied for user: 'paul@localhost'(Using password: NO))
```

Acest rezultat arată că nu a fost stabilită nici o conexiune şi vă permite să aflaţi motivul. De asemenea, înseamnă că nici primul nostru program, `client1`, nu a reuşit să se conecteze la server! (La urma urmelor, `client1` a folosit aceiaşi parametri de conexiune.) La momentul respectiv nu am ştiut acest lucru, deoarece `client1` nu s-a deranjat să verifice apariţia erorilor. `client2` execută această verificare, deci ne poate anunţa dacă se întâmplă ceva grav. De aceea trebuie întotdeauna să testaţi valorile returnate de funcţiile API.

Răspunsurile la întrebările din lista de corespondenţă pentru MySQL au frecvent legătură cu verificarea apariţiei erorilor. Întrebările caracteristice sunt „De ce programul meu cade

atunci când emite această interogare?" sau „Cum se face că interogarea mea nu returnează nici un rezultat?". În multe situații, programul în chestiune nu a verificat dacă acea conexiune a fost sau nu stabilită cu succes înainte de a emite interogarea, respectiv nu a verificat pentru a se asigura că serverul a executat cu succes interogarea înainte de a încerca să regăsească rezultatele. Nu faceți greșeala de a crede că fiecare apel la biblioteca client reușește.

Celelalte exemple din acest capitol execută verificarea apariției erorilor, iar dumneavoastră trebuie să procedați la fel. S-ar putea să vi se pară mai multă muncă, dar pe termen lung este de fapt mai puțină, deoarece vă pierdeți mai puțin timp cu depistarea problemelor subtile. Voi folosi aceeași procedură de verificare a apariției erorilor în capitolele 7, „Interfața API pentru Perl DBI", respectiv 8, „Interfața API pentru PHP".

Acum, să presupunem că ați primit un mesaj `Access denied` când ați rulat programul `client2`. Cum puteți remedia problema? O posibilitate este de a modifica liniile `#define` care conțin numele gazdei, numele utilizatorului și parola, inserând valori care vă permit să obțineți accesul la serverul dumneavoastră. Acest procedeu poate fi benefic, în sensul că măcar veți reuși să stabiliți o conexiune. Dar valorile sunt în continuare codate hard în programul dumneavoastră. Sunt împotriva acestei metode, mai ales în ceea ce privește valoarea parolei. S-ar putea să credeți că parola devine ascunsă atunci când compilați programul în formă binară, dar nu mai este ascunsă deloc dacă cineva poate rula `strings` în programul dumneavoastră. (Ca să nu mai vorbim de faptul că orice persoană cu acces de citire la fișierul dumneavoastră sursă poate obține parola fără să se ostenească deloc.)

Vom aborda problema accesului în secțiunea „Client 4 – Obținerea parametrilor de conexiune la rulare". Mai întâi, aș dori să vă prezint alte metode de scriere a programului de conexiune.

Client 3 – Imprimarea unui caracter modular asupra programului de conexiune

Pentru cel de-al treilea client, `client3`, vom crea un program de conectare și deconectare modular, prin încapsularea sa în funcțiile `do_connect()`, respectiv `do_disconnect()`, care pot fi folosite cu ușurință de mai multe programe client. Această metodă oferă o alternativă la înglobarea literală a programului de conectare în funcția dumneavoastră `main()`. Oricum, este o idee bună pentru orice program care nu se modifică de la o aplicație la alta. Inserați programul într-o funcție accesibilă din mai multe programe, în loc de a-l scrie în fiecare program. Dacă remediați o hibă sau dacă aduceți funcției o îmbunătățire, puteți face modificarea o singură dată, iar toate programele care folosesc funcția vor putea fi remediate sau vor putea beneficia de îmbunătățirea adusă printr-o simplă recompilare. De asemenea, unele programe client sunt scrise de așa manieră încât se pot conecta și deconecta de mai multe ori pe durata execuției lor. Este mult mai simplu să scrieți un asemenea client, dacă imprimați programului un caracter modular prin inserția mecanismelor de pornire și oprire în interiorul funcțiilor de conectare și deconectare.

Strategia de încapsulare funcționează astfel:

1. Divizați programul comun în funcții container într-un fișier sursă separat, denumit `common.c`.

2. Furnizați un fișier antet, `common.h`, care să conțină prototipuri ale rutinelor comune.
3. Includeți fișierul `common.h` în fișierele sursă ale clienților care folosesc rutine comune.
4. Compilați fișierele sursă comune într-un fișier obiect.
5. Legați acel fișier obiect comun la programul dumneavoastră client.

Având în vedere această strategie, să construim funcțiile `do_connect()` și `do_disconnect()`.

Funcția `do_connect()` înlocuiește apelurile la funcțiile `mysql_init()` și `mysql_real_connect()`, precum și programul de afișare a erorilor. Puteți denumi funcția chiar `mysql_real_connect()`, cu deosebirea că nu-i transferați nici o variabilă de tratare a conexiunii. În schimb, `do_connect()` alocă și inițializează singură variabila, iar apoi returnează un pointer spre aceasta după conectare. Dacă `do_connect()` eșuează, afișează un mesaj de eroare și returnează `NULL`. (Astfel, orice program care apelează funcția `do_connect()` și obține o valoare returnată `NULL` își poate încheia pur și simplu execuția, fără a se mai deranja să afișeze un mesaj.)

Funcția `do_disconnect()` preia un pointer spre variabila de tratare a conexiunii și apelează funcția `mysql_close()`.

Iată liniile de program ale fișierului `common.c`:

```
#include <stdio.h>
#include <mysql.h>
#include "common.h"

MYSQL *
do_connect(char *host_name, char *user_name, char *password,
           char *db_name, unsigned int port_num,
           char *socket_name, unsigned int flags)
{
MYSQL *conn        /* pointer spre variabila de tratare a conexiunii */

  conn = mysql_init (NULL);   /* aloca, initializeaza variabila de
                                 tratare a conexiunii */
  if (conn == NULL)
  {
    fprintf (stderr, "mysql_init() failed\n");
    return (NULL);
  }
  if (mysql_real_connect (conn, host_name, user_name, password, db_name,
    port_num, socket_name, flags) == NULL)
  {
    fprintf (stderr, "mysql_real_connect() failed:\nError %u (%s)\n",
                mysql_errno(conn), mysql_error(conn));
    return(NULL);
  }
  return(conn)               /* conexiunea a fost stabilita */
}
```

Continuare

Continuare

```
void
do_disconnect (MYSQL *conn)
{
  mysql_close(conn);
}
```

Fișierul `common.h` declară prototipurile pentru rutinele din fișierul `common.c`:

```
MYSQL *
do_connect(char *host_name, char *user_name, char *password,
           char *db_name, unsigned int port_num,
           char *socket_name, unsigned int flags);
void
do_disconnect (MYSQL *conn);
```

Pentru a obține accesul la rutinele comune, includeți fișierul `common.h` în fișierele dumneavoastră sursă. Rețineți că fișierul `common.c` include și fișierul `common.h`. Astfel, dacă definițiile de funcții din `common.c` nu corespund declarațiilor din fișierul antet, veți primi imediat un avertisment de la compilator. De asemenea, dacă modificați o secvență de apelare din `common.c` fără a face schimbările de rigoare în fișierul `common.h`, atunci când recompilați `common.c` veți primi un avertisment de la compilator.

Este absolut normal să vă întrebați de ce s-a inventat o funcție container, și anume `do_disconnect()`, care execută atât de puține operații. Este adevărat că `do_disconnect()` și `mysql_close()` sunt echivalente. Dar să presupunem că, la un moment ulterior, doriți să faceți o oarecare curățenie suplimentară de fiecare dată când vă deconectați. Prin apelarea unei funcții container asupra căreia aveți un control total, puteți modifica această funcție astfel încât să execute operațiile pe care le doriți, iar modificările intră în funcțiune în același mod pentru toate operațiile de deconectare pe care le executați. Nu puteți face acest lucru dacă invocați direct funcția `mysql_close()`.

Anterior, am afirmat că este benefică „modularizarea“ programelor frecvent folosite prin încapsularea lor într-o funcție care poate fi folosită de mai multe programe, respectiv din mai multe locuri din interiorul unui singur program. Paragraful precedent a oferit un motiv pentru această operație, iar următoarele două exemple vor furniza justificări suplimentare.

- **Exemplul 1.** În versiunile MySQL anterioare seriei 3.22, apelul la funcția `mysql_real_connect()` era ușor diferit de varianta actuală: nu exista nici un parametru care să includă numele bazei de date. Dacă doriți să folosiți funcția `do_connect()` cu o bibliotecă client MySQL mai veche, acest lucru nu este posibil. Totuși, este posibilă modificarea funcției `do_connect()` astfel încât să poată funcționa cu instalări ale sistemului MySQL anterioare versiunii 3.22. Aceasta înseamnă că, prin modificarea funcției `do_connect()`, puteți mări nivelul de portabilitate al tuturor programelor care o folosesc. Dacă înglobați literal programul de conectare în fiecare client, trebuie să modificați fiecare client în parte.

  Pentru a remedia funcția `do_connect()` astfel încât să poată lucra cu forma mai veche a funcției `mysql_real_connect()`, folosiți macroinstrucțiunea `MYSQL_VERSION_ID`, care conține numărul curent al versiunii MySQL. Funcția `do_connect()` modificată testează valoarea parametrului `MYSQL_VERSION_ID` și folosește forma corespunzătoare a funcției `mysql_real_connect()`:


```
MYSQL *
do_connect(char *host_name, char *user_name, char *password,
     char *db_name, unsigned int port_num,
     char *socket_name, unsigned int flags)
{
MYSQL *conn /* pointer spre variabila de tratare a conexiunii */

  conn = mysql_init (NULL);  /* aloca, initializeaza variabila de
                                tratare a conexiunii */
  if (conn == NULL)
  {
    fprintf (stderr, "mysql_init() failed\n");
    return (NULL);
  }
#if defined(MYSQL_VERSION_ID) && MYSQL_VERSION_ID >= 32200
    /* versiunea 3.22 si versiunile ulterioare */
  if (mysql_real_connect (conn, host_name, user_name, password,
    db_name, port_num, socket_name, flags) == NULL)
  {
    fprintf (stderr, "mysql_real_connect() failed:\nError %u (%s)\n",
         mysql_errno(conn), mysql_error(conn));
  return(NULL);
}
#else              /* pentru MySQL anterior versiunii 3.22 */
  if (mysql_real_connect (conn, host_name, user_name, password,
    port_num, socket_name, flags) == NULL)
  {
    fprintf (stderr, "mysql_real_connect() failed:\nError %u (%s)\n",
         mysql_errno(conn), mysql_error(conn));
    return(NULL);
  }
  if (db_name != NULL) /* simularea efectului parametrului db_name */
  {
    if (mysql_select_db (conn, db_name) != 0)
    {
        fprintf (stderr, "mysql_select_db() failed:\nError %u
          (%s)\n", mysql_errno(conn), mysql_error(conn));
        mysql_close(conn);
        return(NULL);
    }
  }
#endif
  return (conn);  /* conexiunea a fost stabilita */
}
```

Versiunea modificată a funcţiei `do_connect()` este similară cu versiunea anterioară din toate punctele de vedere, cu două excepţii:

- Nu transferă un parametru `db_name` formei mai vechi a funcţiei `mysql_real_connect()`, deoarece versiunea respectivă nu dispune de un atare parametru.
- Dacă numele bazei de date nu este `NULL`, funcţia `do_connect()` apelează funcţia `mysql_select_db()` pentru a transforma baza de date denumită în bază de date curentă. (Acest procedeu simulează efectul parametrului `db_name`, care lipseşte.) Dacă baza de date nu poate fi selectată, `do_connect()` afişează un mesaj de eroare, închide conexiunea şi returnează `NULL` pentru a indica eşecul operaţiei.

■ **Exemplul 2.** Acest exemplu se bazează pe modificările aduse funcţiei `do_connect()` în primul exemplu. Aceste modificări au ca rezultat trei seturi de apeluri la funcţiile de eroare `mysql_errno()` şi `mysql_error()` şi este chiar obositor să scrii funcţiile respective de fiecare dată când programul trebuie să „protesteze" la apariţia unei probleme. De asemenea, programul de afişare a erorii este agresiv din punct de vedere vizual şi dificil de citit. Este mai uşor să citiţi ceva de genul acesta:

```
print_error(conn, "mysql_real_connect() failed");
```

Deci, haideţi să încapsulăm scrierea erorilor într-o funcţie `print_error()`. Putem scrie funcţia astfel încât să efectueze o anumită operaţie chiar şi în situaţia când `conn` este `NULL`. Astfel, putem folosi `print_error()` dacă apelul la funcţia `mysql_init()` eşuează şi nu avem o combinaţie de apeluri (unele pentru `fprintf()` şi altele pentru `print_error()`).

Parcă aud pe cineva din spate care obiectează: „Păi nu sunteţi obligat să apelaţi ambele funcţii de eroare de fiecare dată când trebuie să raportaţi o eroare, deci intenţionat vă faceţi programul dificil de citit, pentru ca exemplul dumneavoastră cu încapsularea să arate mai bine. Şi de fapt nici măcar nu veţi scrie tot programul de afişare a erorilor; îl scrieţi o singură dată şi apoi folosiţi copierea şi lipirea dacă mai aveţi nevoie de el." Acestea sunt observaţii corecte, la care voi răspunde astfel:

- Chiar dacă folosiţi copierea şi lipirea, aceste operaţii sunt mai simplu de efectuat cu secţiuni mai scurte de program.
- Indiferent dacă preferaţi sau nu să invocaţi ambele funcţii de eroare la fiecare raportare a unei erori, scrierea integrală a programului de raportare a erorilor în „varianta lungă" duce la tentaţia de a folosi scurtături şi de a fi inconsecvent la raportarea erorilor. Plasarea codului de raportare a erorilor într-o funcţie container care este simplu de invocat atenuează această tentaţie şi îmbunătăţeşte consecvenţa programului.
- Dacă vă decideţi vreodată să modificaţi formatul mesajelor dumneavoastră de eroare, este mult mai uşor dacă trebuie să efectuaţi modificarea într-un singur loc, decât în tot programul. Sau, dacă vă decideţi să scrieţi mesajele de eroare într-un fişier jurnal în loc de (sau în afară de) a le scrie în `stderr`, este mai simplu dacă trebuie să modificaţi numai funcţia `print_error()`. Această metodă este mai puţin expusă la erori şi, din nou, reduce tentaţia de a face numai o jumătate de treabă şi de a fi inconsecvent.

- Dacă folosiţi un utilitar de depanare atunci când vă testaţi programele, inserţia unui punct de întrerupere în funcţia de raportare a erorilor este o modalitate convenabilă de a determina programul să se întrerupă atunci când depanatorul detectează o condiţie de eroare.

Iată funcţia noastră `print_error()` de afişare a erorilor:

```
void
print_error(MYSQL *conn, char *message)
{
   fprintf (stderr, "%s\n", message);
   if (conn != NULL)
   {
      fprintf (stderr, Error %u (%s)\n", mysql_errno(conn),
               mysql_error(conn));
   }
}
```

Funcţia `print_error()` se află în fişierul `common.c`, deci vom adăuga un prototip al acesteia în fişierul `common.h`:

```
void
print_error(MYSQL *conn, char *message);
```

Acum, funcţia `do_connect()` poate fi modificată pentru a folosi funcţia `print_error()`:

```
MYSQL *
do_connect(char *host_name, char *user_name, char *password, char *db_name,
      unsigned int port_num, char *socket_name, unsigned int flags)
{
MYSQL *conn /* pointer spre variabila de tratare a conexiunii */

   conn = mysql_init (NULL);   /* aloca, initializeaza variabila
                                  de tratare a conexiunii */
   if (conn == NULL)
   {
      print_error(NULL, "mysql_init()failed (probably out of memory)");
      return (NULL);
   }
#if defined(MYSQL_VERSION_ID) && MYSQL_VERSION_ID >= 32200
            /* versiunea 3.22 si versiunile ulterioare */
   if (mysql_real_connect (conn, host_name, user_name, password,
      db_name, port_num, socket_name, flags) == NULL)
   {
      print_error(conn, "mysql_real_connect() failed");
      return(NULL);
   }
#else       /* pentru MySQL anterior versiunii 3.22 */
```


Continuare

Continuare

```
    if (mysql_real_connect (conn, host_name, user_name, password,
        port_num, socket_name, flags) == NULL)
    {
        print_error(conn, "mysql_real_connect() failed");
        return(NULL);
    }
    if (db_name != NULL) /* simularea efectului parametrului db_name */
    {
        if (mysql_select_db (conn, db_name) != 0)
        {
            print_error(conn, "mysql_select_db() failed");
            mysql_close(conn);
            return(NULL);
        }
    }
#endif
    return (conn);      /* conexiunea a fost stabilita */
}
```

Fișierul nostru sursă principal, `client3.c`, este asemănător cu `client2.c`, dar au fost eliminate toate liniile de program de conexiune și întrerupere a conexiunii, fiind înlocuite cu apeluri la funcțiile container. Deci, fișierul sursă se prezintă astfel:

```
/* client3.c */

#include <stdio.h>
#include <mysql.h>
#include "common.h"

#define def_host_name NULL    /* gazda la care se va stabili conexiu-
                                 nea(valoare prestabilita = localhost) */
#define def_user_name NULL    /* nume utilizator (valoare prestabilita
                                 = numele dumneavoastra de deschidere
                                 a sesiunii de lucru) */
#define def_password NULL     /* parola (valoare prestabilita =
                                 nici una) */
#define def_db_name NULL      /* baza de date de utilizat (valoare
                                 prestabilita = nici una) */

MYSQL *conn;                  /* pointer spre variabila de tratare a
                                 conexiunii */

int
main (int argc, char *argv[])
{
    conn = do_connect(def_host_name, def_user_name, def_password,
                      def_db_name, def_port_num, def_socket_name, 0);
    if (conn == NULL)
        exit(1);

    /* aici are loc activitatea aplicatiei */

    do_disconnect(conn);
    exit(0);
}
```


Client 4 – Obținerea parametrilor de conexiune la rulare

Acum, când dispunem de un program de conexiune ușor de modificat și „blindat" în cazul apariției erorilor, suntem pregătiți să aflăm cum putem face lucruri mai inteligente decât să folosim parametri de conexiune `NULL`, ca de exemplu să permitem utilizatorului să specifice aceste valori la rulare.

Clientul anterior, `client3`, continuă să prezinte un dezavantaj semnificativ, în sensul că parametrii de conexiune sunt codați în program. Pentru a modifica oricare dintre aceste valori, trebuie să editați fișierul sursă și să-l recompilați, ceea ce nu este foarte convenabil, mai ales dacă doriți să puneți programul la dispoziția altor persoane.

O modalitate frecvent folosită de specificare a parametrilor de conexiune la rulare este de a folosi opțiunile din linia de comandă. Programele din distribuția sistemului MySQL acceptă parametrii de conexiune într-una din cele două forme specificate în tabelul 6.1.

Tabelul 6.1 Opțiuni standard din linia de comandă pentru MySQL

Parametru	Formă scurtă	Formă lungă
Numele gazdei	`-h` *nume_gazda*	`--host=`*nume_gazda*
Numele de utilizator	`-u` *nume_utilizator*	`--user=`*nume_utilizator*
Parola	`-p` sau `-p`*parola_dv*	`--password` sau `--password=`*parola_dv*
Numărul portului	`-P` *numar_port*	`--port=`*numar_port*
Numele soclului	`-S` *nume_soclu*	`--socket=`*nume_soclu*

Pentru consecvența cu programele client MySQL standard, clientul nostru va accepta aceleași formate, ceea ce este simplu de realizat, deoarece biblioteca client include o funcție pentru analiza opțiunilor.

În plus, clientul nostru va putea să extragă informațiile din fișierele cu opțiuni. Aceasta vă permite să plasați parametrii de conexiune în fișierul `~/.my.cnf` (adică fișierul `.my.cnf` din catalogul dumneavoastră de bază), astfel încât să nu fie necesară specificarea lor în linia de comandă. Biblioteca client facilitează căutarea fișierelor cu opțiuni MySQL și extragerea din acestea a tuturor valorilor relevante. Prin adăugarea a numai câteva linii

în programul dumneavoastră, puteți determina programul să recunoască fișierele cu opțiuni și nu trebuie să reinventați roata prin scrierea propriilor dumneavoastră programe pentru aceasta. Sintaxa fișierelor cu opțiuni este descrisă în Anexa E, „Referință de programe MySQL".

Accesul la conținutul fișierelor cu opțiuni

Pentru a citi fișierele cu opțiuni în căutarea valorilor parametrilor de conexiune, folosiți funcția `load-defaults()`. Această funcție caută fișierele cu opțiuni, le analizează conținutul pentru a descoperi orice grupuri de opțiuni care vă interesează și rescrie vectorul cu argumente al programului dumneavoastră (tabloul `argv[]`) pentru a insera informațiile din aceste grupuri sub formă de opțiuni ale liniei de comandă la începutul tabloului `argv[]`. Astfel, opțiunile apar ca și cum ar fi fost specificate în linia de comandă. Ca atare, când analizați opțiunile comenzii, obțineți parametrii de conexiune ca parte a ciclului normal de analiză a opțiunilor. Opțiunile sunt adăugate la începutul tabloului `argv[]` și nu la sfârșit, astfel încât, dacă parametrii de conexiune sunt într-adevăr specificați în linia de comandă, aceștia să apară mai târziu (și implicit să redefinească) orice opțiuni adăugate de funcția `load_defaults()`.

Iată un mic program, `show_argv`, care prezintă modul de utilizare a funcției `load_defaults()` și care ilustrează modificarea vectorului cu argumente prin acest procedeu:

```
/* show_argv.c */

#include <stdio.h>
#include <mysql.h>

char *groups[] = { "client", NULL };

int
main(int argc, char *argv[])
{
int i;

   my_init();

   printf("Vector cu argumente original:\n");
   for(i = 0; i < argc; i++)
      printf("arg %d: %s\n", i, argv[i]);

   load_defaults("my", groups, &argc, &argv);

   printf("Vector cu argumente modificat:\n");
   for(i = 0; i < argc; i++)
      printf("arg %d: %s\n", i, argv[i]);

   exit(0);
}
```


Programul de prelucrare a fișierului cu opțiuni implică următoarele:

- `groups[]` este un tablou șir de caractere care indică grupurile din fișierele cu opțiuni care vă interesează. Pentru programele client, specificați întotdeauna cel puțin menționea `"client"` (pentru grupul `[client]`). Ultimul element al tabloului trebuie să fie `NULL`.
- `my_init()` este o rutină de inițializare care execută unele operații de pornire impuse de funcția `load_defaults()`.
- Funcția `load_defaults()` preia patru argumente: prefixul fișierelor dumneavoastră cu opțiuni (acesta trebuie să fie întotdeauna `"my"`), tabloul care menționează grupurile de opțiuni care vă interesează, respectiv adresa numărului de argumente și a vectorului de argumente ale programului dumneavoastră. Nu transferați valorile numărului de argumente și ale vectorului; transmiteți în schimb adresele lor, deoarece funcția `load_defaults()` trebuie să le modifice valorile. Rețineți, mai ales, că deși `argv` este un pointer, trebuie să transferați `&argv`, adresa pointerului respectiv.

Funcția `show_argv` își afișează argumentele de două ori: prima dată așa cum le-ați specificat în linia de comandă, apoi în urma modificărilor efectuate de `load_defaults()`. Pentru a vedea efectele funcției `load_defaults()`, asigurați-vă că aveți un fișier `.my.cnf` în catalogul dumneavoastră de bază, cu unii parametri specificați pentru grupul `[client]`. Să presupunem că fișierul `.my.cnf` se prezintă astfel:

```
[client]
user=paul
password=secret
host=o_gazda
```

În această situație, prin executarea programului `show_argv` se obține următorul rezultat:

```
% show_argv a b
Vector cu argumente original:
arg 0: show_argv
arg 1: a
arg 2: b
Vector cu argumente modificat:
arg 0: show_argv
arg 1: --user=paul
arg 2: --password=secret
arg 3: --host=o_gazda
arg 4: a
arg 5: b
```

Este posibil ca între datele de ieșire ale programului `show_argv` să vedeți unele opțiuni care nu se găseau nici în linia de comandă, nici în fișierul dumneavoastră `~/.my.cnf`. În acest caz, opțiunile respective au fost probabil specificate într-un fișier cu opțiuni la nivel de sistem. De fapt, funcția `load_defaults()` caută fișierele `/etc/my.cnf` și `my.cnf` din catalogul de date MySQL înainte de a citi fișierul `.my.cnf` din catalogul dumneavoastră de bază. (În Windows, funcția `load_defaults()` caută fișierele `C:\my.cnf`, `C:\mysql\data\my.cnf`, respectiv fișierul `my.ini` din catalogul \Windows\System).

Programele client care folosesc funcția `load_defaults()` specifică aproape întotdeauna "client" în lista cu grupuri de opțiuni (pentru a putea obține toate valorile client generale din fișierele cu opțiuni), dar puteți cere și valori care sunt specifice propriului dumneavoastră program. Pur și simplu înlocuiți instrucțiunea

```
char *groups[] = { "client", NULL };
```

cu următoarea linie de program:

```
char *groups[] = { "show_argv", "client", NULL };
```

Apoi, puteți adăuga un grup `[show_argv]` la fișierul dumneavoastră `~/.my.cnf`:

```
[client]
user=paul
password=secret
host=o_gazda

[show_argv]
host=alta_gazda
```

În urma acestor modificări, o nouă invocare a programului `show_argv` va avea un alt rezultat, după cum urmează:

```
% show_argv a b
Vector cu argumente original:
arg 0: show_argv
arg 1: a
arg 2: b
Vector cu argumente modificat:
arg 0: show_argv
arg 1: --user=paul
arg 2: --password=secret
arg 3: --host=o_gazda
arg 4: --host=alta_gazda
arg 5: a
arg 6: b
```

Ordinea în care apar valorile opțiunilor în tabloul cu argumente este determinată de ordinea în care acestea sunt menționate în fișierul dumneavoastră cu opțiuni, nu în ordinea în care grupurile cu opțiuni sunt menționate în tabloul `groups[]`. Aceasta înseamnă că probabil veți dori să specificați grupuri specifice programelor după grupul `[client]` din fișierul dumneavoastră cu opțiuni. Astfel, dacă specificați o opțiune în ambele grupuri, valoarea specifică programului va avea prioritate. Puteți vedea acest lucru în exemplul prezentat anterior: opțiunea `host` a fost specificată atât în grupul `[client]`, cât și în grupul `[show_argv]` dar, deoarece grupul `[show_argv]` apare ultimul în fișierul cu opțiuni, valoarea sa `host` apare mai târziu în vectorul cu argumente și are prioritate.

Funcția `load_defaults()` nu selectează valori din parametrii dumneavoastră de mediu. Dacă doriți să folosiți valorile unor variabile de mediu precum `MYSQL_TCP_PORT` sau `MYSQL_UNIX_PORT`, trebuie să vă ocupați personal de acest lucru prin intermediul funcției `getenv()`. Nu voi adăuga această funcționalitate la clienții noștri, dar iată un exemplu de verificare a valorilor a două dintre variabilele de mediu standard legate de MySQL:


```
extern char *getenv();
char *p;
int port_num;
char *socket_name;

if ((p = getenv("MYSQL_TCP_PORT") != NULL)
    port_num = atoi (p);
if ((p = getenv("MYSQL_UNIX_PORT") != NULL)
    socket_name = p;
```

În cazul clienților MySQL standard, valorile variabilelor de mediu au o prioritate mai redusă decât valorile specificate în fișierele cu opțiuni sau în linia de comandă. Dacă verificați variabilele membru și doriți să respectați convenția respectivă, verificați variabilele de mediu înainte, nu după apelarea funcției `load_defaults()` sau prelucrarea opțiunilor din linia de comandă.

Analiza argumentelor din linia de comandă

În acest moment, putem prelua toți parametrii de conexiune în vectorul cu argumente, dar avem nevoie de o modalitate de analiză a vectorului. Funcția `getopt()` are exact această destinație.

Funcția `getopt_long()` este încorporată în biblioteca client MySQL, deci puteți avea acces la aceasta ori de câte ori stabiliți legături cu funcții din biblioteca respectivă. În fișierul dumneavoastră sursă, trebuie să includeți fișierul antet `getopt.h`. Puteți copia acest fișier antet din catalogul `include` al distribuției sursă MySQL în catalogul în care vă dezvoltați programul client.

Funcția `load_defaults()` și securitatea

Poate vă puneți întrebări cu privire la implicațiile legate de „spionarea" proceselor pe care le poate avea solicitarea ca funcția `load_defaults()` să insereze textul parolelor în lista dumneavoastră cu argumente, deoarece programe precum `ps` pot afișa listele cu argumente pentru procese arbitrare. Nu este nici o problemă, deoarece `ps` afișează conținutul original al tabloului `argv[]`. Toate argumentele de tip parolă create de funcția `load_defaults()` indică spre o regiune pe care funcția respectivă o alocă pentru sine. Acea regiune nu face parte din vectorul original, deci programul `ps` nu o vede niciodată.

Pe de altă parte, o parolă care este specificată în linia de comandă *apare* în `ps` dacă dumneavoastră nu vă îngrijiți să o ștergeți. Secțiunea „Analiza argumentelor din linia de comandă" vă arată cum să procedați.

Programul următor, `show_param`, folosește funcția `load_defaults()` pentru citirea fișierelor cu opțiuni, apoi apelează funcția `getopt_long()` pentru analiza vectorului cu argumente. `show_param` arată ce se întâmplă în fiecare fază a prelucrării argumentelor, prin efectuarea următoarelor acțiuni:

1. Configurează valorile prestabilite pentru numele gazdei, numele de utilizator și parola.
2. Afișează valorile originale ale parametrilor de conexiune și valorile din vectorul cu argumente.

3. Apelează funcția `load_defaults()` pentru a rescrie vectorul cu argumente astfel încât acesta să reflecte conținutul fișierului cu opțiuni, după care afișează vectorul rezultant.
4. Apelează funcția `getopt_long()` pentru prelucrarea vectorului cu argumente, apoi afișează valorile rezultante ale parametrilor și ceea ce a mai rămas în vectorul cu argumente.

`show_param` vă permite să exersați diferite modalități de specificare a parametrilor de conexiune (situați în fișierele cu opțiuni sau în linia de comandă) și să vedeți rezultatele prin afișarea valorilor care vor fi folosite pentru stabilirea unei conexiuni. `show_param` este util pentru a vă face o idee privind ceea ce se va întâmpla în următorul nostru program client, atunci când corelăm acest program de prelucrare a parametrilor cu funcția noastră de conectare `do_connect()`.

Iată cum se prezintă fișierul `show_param.c`:

```
/* show_param.c */

#include <stdio.h>
#include <stdlib.h> /* necesar pentru atoi() */
#include "getopt.h"

char *groups[] = { "client", NULL };

struct option long_options[] =
{
    {"host",          required_argument, NULL, 'h'},
    {"user",          required_argument, NULL, 'u'},
    {"password",      optional_argument, NULL, 'p'},
    {"port",          required_argument, NULL, 'P'},
    {"socket",        required_argument, NULL, 'S'},
    { 0, 0, 0, 0 }
};

int
main (int argc, char *argv[])
{
char *host_name = NULL;
char *user_name = NULL;
char *password = NULL;
unsigned int port_num = 0;
char *socket_name = NULL;
int i;
int c, option_index;

my_init();

printf ("Parametrii originali ai conexiunii:\n")
```



```
printf ("host name: %s\n", host_name ? host_name : "(null)");
printf ("user name: %s\n", user_name ? user_name : "(null)");
printf ("password: %s\n", password ? password : "(null)");
printf ("port number: %u\n", port_num);
printf ("socket name: %s\n", socket_name ? socket_name : "(null)");

printf ("Vector cu argumente original:\n");
for (i = 0; i < argc; i++)
   printf ("arg %d: %s\n", i, argv[i]);

load_defaults("my", groups, &argc, &argv);

printf("Vector cu argumente modificat dupa load_defaults():\n");
for (i = 0; i < argc; i++)
   printf("arg %d: %s\n", i, argv[i]);

while ((c = getopt_long(argc, argv, "h:p::u:P:S",
      long_options, &option_index)) != EOF)
{
   switch (c)
   {
   case 'h':
      host_name = optarg;
      break;
   case 'u':
      user_name = optarg;
      break;
   case 'p':
      password = optarg;
      break;
   case 'P':
      port_num = (unsigned int) atoi (optarg);
      break;
   case 'S':
      socket_name = optarg;
      break;
   }
}

argc -= optind; /* avanseaza dincolo de argumentele */
argv += optind; /* care au fost prelucrate de getopt_long() */

printf ("Parametrii conexiunii dupa getopt_long():\n")
printf ("host name: %s\n", host_name ? host_name : "(null)");
```

Continuare

Continuare

```
printf ("user name: %s\n", user_name ? user_name : "(null)");
printf ("password: %s\n", password ? password : "(null)");
printf ("port number: %u\n", port_num);
printf ("socket name: %s\n", socket_name ? socket_name : "(null)");

printf ("Vector cu argumente dupa getopt_long():\n");
for (i = 0; i < argc; i++)
  printf("arg %d: %s\n", i, argv[i]);
exit(0);
}
```

Pentru a prelucra vectorul cu argumente, show_argv folosește funcția getopt_long(), pe care o apelați, în mod caracteristic, într-un ciclu:

```
while ((c = getopt_long(argc, argv, "h:p::u:P:S",
            long_options, &option_index)) != EOF)
{
    /* optiune de proces */
}
```

Primele două argumente ale funcției getopt_long() le constituie numărul de argumente al programului dumneavoastră și vectorul cu argumente. Al treilea argument îl reprezintă literele de opțiuni care doriți să fie recunoscute. Acestea sunt formele cu nume scurt ale opțiunilor programului dumneavoastră. Literele de opțiune pot fi urmate de două puncte, de două puncte dublate (::) sau de nici un asemenea caracter, pentru a arăta că opțiunea trebuie să fie urmată, poate fi urmată, respectiv nu este urmată de o valoare a opțiunii. Cel de-al patrulea argument, long_options, este un pointer spre un tablou cu structuri de opțiune, fiecare din aceste structuri specificând informații pentru o opțiune pe care doriți ca programul dumneavoastră să o înțeleagă. Rolul său este similar cu acela al șirului de opțiuni din al treilea argument. Cele patru elemente ale fiecărei structuri long_options[] sunt următoarele:

- **Numele lung al opțiunii.**
- **O valoare a opțiunii.** Valoarea poate fi required_argument (argument obligatoriu), optional_argument (argument facultativ) sau no_argument (fără argument), care indică dacă opțiunea trebuie urmată, poate fi urmată, respectiv nu este urmată de o valoare a opțiunii. (Aceste valori au același rol ca și caracterul două puncte, caracterul două puncte dublate și respectiv absența oricărui caracter din al treilea argument, care conține șirul de opțiuni.)
- **Un argument indicator (flag).** Puteți folosi acest argument pentru a stoca un pointer la o variabilă. Dacă se găsește opțiunea, getopt_long() stochează în variabilă valoarea specificată de al patrulea argument. Dacă indicatorul este NULL, getopt_long() configurează variabila optarg astfel încât să indice spre orice valoare care urmează după opțiune și returnează numele scurt al opțiunii. În cazul nostru, tabloul long_options[] specifică NULL pentru toate opțiunile. Astfel, getopt_long() returnează fiecare argument așa cum îl întâlnește astfel încât să poată fi prelucrat în instrucțiunea switch.

- **Numele scurt (dintr-un singur caracter) al opțiunii.** Numele scurte precizate în tabloul long_options[] *trebuie* să corespundă literelor folosite în șirul cu opțiuni pe care îl transferați drept al treilea argument al funcției getopt_long(), în caz contrar programul dumneavoastră fiind incapabil de a prelucra în mod adecvat argumentele din linia de comandă.

Tabloul long_options[] trebuie terminat cu o structură ale cărei elemente sunt toate egale cu 0.

Cel de-al cincilea argument al funcției getopt_long() este un pointer la o variabilă de tip int. getopt_long() stochează în această variabilă indexul structurii long_options[] care corespunde ultimei opțiuni întâlnite. (show_param nu face nimic cu această valoare.)

Rețineți că opțiunea pentru parolă (specificată sub forma --password sau -p) poate lua o valoare opțională. Cu alte cuvinte, o puteți specifica sub forma --password sau --password=*parola_dumneavoastra* dacă folosiți forma de opțiune lungă, respectiv -p sau --p*parola_dumneavoastra* dacă folosiți forma de opțiune scurtă. Natura opțională a valorii parolei este indicată de caracterul :: plasat după litera p din șirul cu opțiuni, precum și de specificația optional_argument din tabloul long_options[]. De regulă, clienții MySQL vă permit să omiteți valoarea parolei în linia de comandă, după care vă solicită această valoare. Acest procedeu vă permite să evitați furnizarea parolei în linia de comandă, ceea ce împiedică pe alții să vă citească parola prin „spionarea" procesului. Când vom scrie următorul client, client4, vom adăuga la acesta logica de verificare a parolei.

Iată un exemplu de invocare a programului show_param și a datelor de ieșire rezultate (presupunând că ~/.my.cnf are același conținut ca în exemplul cu programul show_argv):

```
% show_param -h inca_o_gazda x
Parametri de conexiune originali:
host name: (null)
user name: (null)
password: (null)
port number: 0
socket name: (null)
Vector cu argumente original:
arg 0: show_param
arg 1: -h
arg 2: inca_o_gazda
arg 3: x
Vector cu argumente modificat dupa load_defaults():
arg 0: show_param
arg 1: --user=paul
arg 2: --password=secret
arg 3: --host=o_gazda
arg 4: -h
arg 5: inca_o_gazda
```

Continuare

Continuare

```
arg 6: x
Parametri de conexiune dupa getopt_long():
host name: inca_o_gazda
user name: paul
password: secret
port number : 0
socket name: (null)
Vector cu argumente dupa getopt_long():
arg 0: x
```

Datele de ieșire arată că numele gazdei este selectat din linia de comandă (redefinind valoarea din fișierul cu opțiuni), precum și că numele de utilizator și parola provin de la fișierul cu opțiuni. `getopt_long()` analizează corect opțiunile, indiferent dacă acestea sunt specificate în forma de opțiune scurtă (`-h nume_gazda`) sau în forma de opțiune lungă (`--user=paul`, `--password=secret`).

Acum, să eliminăm liniile de program care nu au decât rolul de a ilustra modul de funcționare a rutinelor de tratare a opțiunilor și să folosim restul programului ca bază pentru un client care se conectează la un server în conformitate cu opțiunile care sunt furnizate într-un fișier cu opțiuni sau în linia de comandă. Fișierul sursă rezultant, `client4.c`, se prezintă astfel:

```
/* client4.c */

#include <stdio.h>
#include <stdlib.h> /* pentru atoi() */
#include <mysql.h>
#include "common.h"
#include "getopt.h"

#define def_host_name NULL    /* gazda la care se va stabili
                                 conexiunea(valoare prestabilita =
                                 localhost) */
#define def_user_name NULL    /* nume utilizator (valoare prestabilita =
                                 numele dumneavoastra de deschidere a
                                 sesiunii de lucru) */
#define def_password NULL     /* parola (valoare prestabilita = nici
                                 una) */
#define def_db_name NULL      /* baza de date de utilizat (valoare
                                 prestabilita = nici una) */

char *groups[] = { "client", NULL };

struct option long_options[] =
{
  {"host",             required_argument, NULL, 'h'},
```



```
  {"user",             required_argument, NULL, 'u'},
  {"password",         optional_argument, NULL, 'p'},
  {"port",             required_argument, NULL, 'P'},
  {"socket",           required_argument, NULL, 'S'},
  { 0, 0, 0, 0 }
};

MYSQL *conn;                   /* pointer spre variabila de tratare a
                                  conexiunii */

int
main (int argc, char *argv[])
{
char *host_name = def_host_name;
char *user_name = def_user_name;
char *password = def_password;
unsigned int port_num = def_port_num;
char *socket_name = def_socket_name;
char *db_name = def_db_name;
char passbuf[100];
int ask_password = 0;
int i;
int c, option_index=0;

   my_init();

   load_defaults("my", groups, &argc, &argv);

   while ((c = getopt_long(argc, argv, "h:p::u:P:S", long_options,
           &option_index)) != EOF)
   {
      switch (c)
      {
      case 'h':
           host_name = optarg;
           break;
      case 'u':
           user_name = optarg;
           break;
      case 'p':
           if (!optarg) /* nu este data nici o valoare */
              ask_password = 1;
           else               /* copiaza parola, sterge originalul */
```

Continuare

Continuare

```
            {
                (void) strncpy (passbuf, optarg,
                                sizeof(passbuf)-1);
                passbuf[sizeof(passbuf)-1] = '\0';
                password = passbuf;
                while (*optarg)
                    *optarg++ = ' ';
            }
            break;
        case 'P':
            port_num = (unsigned int) atoi (optarg);
            break;
        case 'S':
            socket_name = optarg;
            break;
        }
    }

    argc -= optind; /* avanseaza dincolo de argumentele */
    argv += optind; /* prelucrate de getopt_long() */
      if (argc > 0)
    {
      db_name = argv[0];
      --argc; ++argv;;
    }

    if (ask_password)
      password = get_tty_password (NULL);

    conn = do_connect (host_name, user_name, password,
                        db_name, port_num, socket_name, 0);
    if (conn == NULL)
      exit(1);

    /* aici se insereaza codul aplicatiei propriu-zise */

    do_disconnect(conn);
    exit(0);
}
```

În comparaţie cu programele client1, client2 şi client3 pe care le-am creat anterior, client4 efectuează câteva operaţii pe care nu le-am întâlnit până acum:

- Permite specificarea numelui bazei de date în linia de comandă, după opţiunile care sunt analizate de getopt_long(). Această comportare este similară cu aceea a clienţilor standard din distribuţia MySQL.

- Şterge valoarea parolei din vectorul cu argumente, după copierea acesteia. Procedeul are ca scop reducerea ferestrei de timp pe durata căreia o parolă specificată în linia de comandă este vizibilă pentru ps sau pentru alte programe de determinare a stării sistemului. (Fereastra este *minimizată*, nu eliminată. Specificarea parolelor în linia de comandă continuă să reprezinte un pericol pentru securitate.)
- Dacă o opţiune de parolă a fost dată fără o valoare, clientul solicită utilizatorului o parolă folosind funcţia get_tty_password(). Aceasta este o rutină de tip utilitar din biblioteca client, care solicită o parolă fără a o reflecta pe ecran. (Biblioteca client este plină de asemenea bunătăţi. O lectură a sursei programelor client MySQL este instructivă, deoarece puteţi afla informaţii despre aceste rutine şi despre modul de utilizare a lor.) Puteţi întreba: „De ce nu apelaţi pur şi simplu funcţia getpass()?" Răspunsul este că nu toate sistemele dispun de această funcţie; Windows este un asemenea sistem. Funcţia get_tty_password() este portabilă pe mai multe sisteme, deoarece este configurată să se ajusteze la idiosincraziile sistemului.

Programul client4 reacţionează în conformitate cu opţiunile pe care le specificaţi. Pentru a complica situaţia, să presupunem că nu există nici un fişier cu opţiuni. Dacă invocaţi programul client4 fără argumente, acesta se conectează la localhost şi transferă serverului numele dumneavoastră UNIX de deschidere a sesiunii de lucru, fără parolă. Dacă în schimb invocaţi client4 aşa cum s-a arătat mai sus, programul va cere o parolă (nu există nici o valoare a parolei imediat după -p), se conectează la o_gazda şi transferă serverului numele de utilizator un_utilizator şi parola pe care o tastaţi:

```
% client4 -h o_gazda -u un_utilizator -p o_baza_de_date
```

De asemenea, client4 mai transferă funcţiei do_connect() numele bazei de date o_baza_de_date, pentru a o transforma în bază de date curentă. Dacă există un fişier cu opţiuni, conţinutul acestuia este prelucrat şi folosit pentru modificarea în consecinţă a parametrilor de conexiune.

Anterior, am încapsulat programele „într-o veselie", creând funcţii container pentru conectarea la, respectiv deconectarea de la server. Se poate pune întrebarea dacă este necesară inserţia unei logici de analiză a opţiunilor şi într-o funcţie container. Este posibil, cred, dar nu voi proceda astfel. Logica de analizare a opţiunilor nu este la fel de consecventă în diferite programe ca şi programul de conexiune: programele acceptă deseori şi alte opţiuni, în afara opţiunilor standard pe care tocmai le-am analizat, iar programe diferite au tendinţa de a accepta alte seturi diferite de opţiuni suplimentare. Astfel, este dificil de scris o funcţie care să standardizeze ciclul de prelucrare a opţiunilor. De asemenea, spre deosebire de stabilirea conexiunii, pe care un program poate dori să o efectueze de mai multe ori pe durata execuţiei sale (şi care este, implicit, un bun candidat pentru încapsulare), analiza opţiunilor se execută de obicei o singură dată: la începutul programului.

Prin ceea ce am făcut până acum, am realizat o operaţie necesară pentru fiecare client MySQL: conectarea la server folosind parametri adecvaţi. Trebuie să ştiţi să vă conectaţi, desigur. Dar acum ştiţi cum să procedaţi, iar detaliile acestui proces sunt implementate în scheletul programului client (client4.c), deci nu mai trebuie să vă gândiţi la ele. Aceasta înseamnă că vă puteţi concentra asupra a ceea ce vă interesează cu adevărat – capacitatea de a obţine acces la conţinutul bazelor dumneavoastră de date. Toată „acţiunea" aplicaţiei

dumneavoastră se va produce în intervalul dintre apelurile la funcțiile `do_connect()` și `do_disconnect()`, dar ceea ce avem acum servește drept cadru general, pe care îl puteți folosi pentru numeroși clienți diferiți. Pentru a scrie un program nou, procedați astfel:

1. Faceți o copie a programului `client4.c`.
2. Modificați ciclul de prelucrare a opțiunilor, dacă acceptați opțiuni suplimentare, altele decât opțiunile standard pe care le cunoaște programul `client4.c`.
3. Adăugați propriul dumneavoastră program de aplicație specific, plasându-l între apelurile de conectare și de deconectare.

Și ați terminat.

Ideea de a respecta o disciplină a construcției unui schelet al programului client era de a propune ceva ușor de folosit pentru a stabili, respectiv pentru a elimina o conexiune, astfel încât dumneavoastră să vă puteți concentra cu adevărat asupra a ceea ce doriți să faceți. Acum sunteți liberi de a proceda astfel, ca o ilustrare a principiului că disciplina duce la libertate.

Prelucrarea interogărilor

Acum, când știm cum să începem și să încheiem o conversație cu serverul, este momentul să aflăm cum se poate dirija conversația în timpul desfășurării acesteia. Această secțiune vă arată cum se poate „discuta" cu serverul pentru prelucrarea interogărilor.

Fiecare interogare pe care o rulați implică următoarele etape:

1. **Construiți interogarea.** Modalitatea de construcție a interogării depinde de conținutul interogării – mai exact, de faptul dacă aceasta conține sau nu date binare.
2. **Emiteți interogarea trimițând-o la server în vederea executării.**
3. **Prelucrați rezultatul interogării.** Aceasta depinde de tipul interogării emise. De exemplu, o instrucțiune `SELECT` returnează rânduri de date pe care urmează să le prelucrați. O instrucțiune `INSERT` nu procedează astfel.

Un factor care trebuie avut în vedere la construirea interogărilor îl constituie funcția care se va utiliza pentru trimiterea acestora la server. Rutina cu un caracter mai general de emitere a interogărilor este `mysql_real_query()`. Cu această rutină, furnizați interogarea sub forma unui șir numărat (un șir plus o lungime). Trebuie să păstrați evidența lungimii interogărilor dumneavoastră și să o trimiteți funcției `mysql_real_query()` alături de șirul în sine. Deoarece interogarea este un șir numărat, conținutul său poate fi oricare, inclusiv date binare sau octeți `NULL`. Interogarea nu este tratată ca un șir care se termină cu o valoare zero[1].

Cealaltă funcție de emitere a interogărilor, și anume `mysql_query()`, este mai restrictivă în ceea ce privește conținutul șirului interogării, dar deseori este mai simplu de utilizat. Interogările pe care le transmiteți funcției `mysql_query()` trebuie să fie șiruri terminate cu zero, ceea ce înseamnă că nu pot conține octeți zero în textul interogării. (Prezența

[1] Prin „valoare zero" nu trebuie să se înțeleagă cifra 0, ci valoarea echivalentă cu zero pentru tipul de date al șirului respectiv. – N.T.

octeților zero în cadrul interogării determină interpretarea eronată a acesteia ca fiind mai scurtă decât în realitate.) În general vorbind, dacă interogarea dumneavoastră poate conține date binare arbitrare, ar putea conține octeți zero, deci nu trebuie să folosiți `mysql_query()`. Pe de altă parte, când lucrați cu șiruri care se termină în zero, vă puteți permite luxul de a construi interogări folosind funcțiile șir din biblioteca C standard pe care probabil că le cunoașteți deja, precum `strcpy()` și `sprintf()`.

Un alt factor de care trebuie ținut cont în construirea interogărilor este dacă trebuie să efectuați operații de anulare a semnificațiilor unor caractere (escaping). Aceste operații sunt necesare dacă doriți să construiți interogări folosind valori care conțin date binare sau alte caractere problematice, cum sunt ghilimelele și caracterele backslash. Despre acest aspect vom discuta în secțiunea „Codificarea datelor problematice în cadrul interogărilor".

O schemă simplă de tratare a interogărilor se prezintă astfel:

```
if (mysql_query (conn, query) != 0)
{
    /* esec; raporteaza eroarea */
}
else
{
    /* succes; afla care a fost efectul interogarii */
}
```

Atât `mysql_query()` cât și `mysql_real_query()` returnează zero pentru interogările care reușesc, respectiv valori diferite de zero în caz de eșec. A spune că o interogare a „reușit" înseamnă că serverul a acceptat-o ca fiind valabilă și că a fost capabil să o execute. Nu se specifică nimic despre efectul interogării. De exemplu, nu se arată că o interogare `SELECT` a selectat vreun rând sau că o instrucțiune `DELETE` a șters vreun rând. Determinarea efectului real al interogării implică prelucrări suplimentare.

O interogare poate eșua dintr-o diversitate de motive. Unele cauze comune includ următoarele:

- Interogarea conține o eroare de sintaxă.
- Interogarea este incorectă din punct de vedere semantic – de exemplu, o interogare care se referă la o coloană inexistentă a unui tabel.
- Nu aveți suficiente privilegii pentru a avea acces la datele la care se face referire în interogare.

Interogările pot fi grupate în două mari categorii: interogările care nu returnează un rezultat și cele care returnează un rezultat. Interogările pentru instrucțiuni precum `INSERT`, `DELETE` și `UPDATE` se încadrează toate în categoria celor care nu returnează nici un rezultat. Interogările respective nu returnează nici un rând, nici măcar în cazul interogărilor care modifică baza dumneavoastră de date. Singura informație pe care o primiți constă în numărul rândurilor afectate.

Interogările pentru instrucțiuni precum `SELECT` și `SHOW` se încadrează în categoria celor care returnează un rezultat; la urma urmelor, scopul emiterii acestor instrucțiuni este de a primi un rezultat. Setul de rânduri produs de o interogare care returnează date se numește set de

rezultate. Acesta este reprezentat în MySQL de tipul de date `MYSQL_RES`, o structură care conține valorile de date pentru rânduri și, de asemenea, metadate referitoare la valori (cum ar fi numele coloanelor și lungimile valorilor de date). Un set de rezultate vid (adică acela care conține zero rânduri) este diferit de noțiunea „fără rezultat".

Tratarea interogărilor care nu returnează un set de rezultate

Pentru a prelucra o interogare care nu returnează un set de rezultate, emiteți interogarea folosind `mysql_query()` sau `mysql_real_query()`. Dacă interogarea reușește, puteți afla care a fost numărul rândurilor inserate, șterse sau actualizate apelând funcția `mysql_affected_rows()`.

Exemplul următor prezintă modul de tratare a unei interogări care nu returnează nici un set de rezultate:

```
if (mysql_query (conn, "INSERT INTO tabel SET nume = 'Numele meu'") != 0)
{
    print_error ("Instructiunea INSERT a esuat");
}
else
{
    printf ("Instructiunea INSERT a reusit: %lu randuri afectate\n",
            (unsigned long) mysql_affected_rows (conn));
}
```

Observați cum rezultatul funcției `mysql_affected_rows()` este convertit la tipul `unsigned long` în vederea afișării. Această funcție returnează o valoare de tipul `my_ulonglong`, dar încercarea de afișare direct pe ecran a unei valori de acest tip nu reușește pe unele sisteme. (De exemplu, am observat că reușește sub FreeBSD, dar eșuează sub Solaris.) Conversia valorii la tipul `unsigned long` și utilizarea unui format de afișare ca `%lu` rezolvă problema. Aceleași considerații se aplică și la orice alte funcții care returnează valori de tipul `my_ulonglong`, precum `mysql_num_rows()` și `mysql_insert_id()`. Dacă doriți ca programele dumneavoastră client să fie portabile pe mai multe sisteme, rețineți acest lucru.

Funcția `mysql_affected_rows()` returnează numărul rândurilor afectate de interogare, dar semnificația noțiunii de „rânduri afectate" depinde de tipul interogării. Pentru instrucțiunile `INSERT`, `REPLACE` sau `DELETE`, semnifică numărul rândurilor inserate, înlocuite, respectiv șterse. Pentru `UPDATE`, este vorba de numărul rândurilor actualizate, ceea ce înseamnă numărul rândurilor pe care MySQL le-a modificat efectiv. MySQL nu actualizează un rând în cazul în care conținutul său este același cu acela al datelor de actualizare. Aceasta înseamnă că, deși un rând poate fi selectat pentru actualizare (de către clauza `WHERE` a instrucțiunii `UPDATE`), este posibil ca rândul respectiv să nu fie modificat.

De fapt, semnificația noțiunii de „rânduri afectate" pentru instrucțiunea `UPDATE` este oarecum controversată, deoarece unii vor ca aceasta să însemne „rânduri corespunzătoare" – adică numărul rândurilor selectate pentru actualizare, chiar dacă operația de actualizare nu modifică valorile pe care le conțin acestea. Dacă aplicația dumneavoastră necesită o atare semnificație, puteți obține această comportare prin solicitare directă în momentul

conectării la server. Transferați funcției `mysql_real_connect()` o valoare `flags` egală cu `CLIENT_FOUND_ROWS`. Aceeași valoare poate fi transmisă ca argument `flags` și funcției `do_connect()`; aceasta va trimite valoarea mai departe funcției `mysql_real_connect()`.

Tratarea interogărilor care returnează un set de rezultate

Interogările care returnează date le grupează pe acestea într-un set de rezultate, pe care îl manipulați după emiterea interogării prin apelarea uneia din funcțiile `mysql_query()` sau `mysql_real_query()`. Este important de reținut că, în MySQL, `SELECT` nu este singura instrucțiune care returnează rânduri. La fel procedează și `SHOW`, `DESCRIBE` și `EXPLAIN`. Pentru toate aceste instrucțiuni, după emiterea interogării trebuie să efectuați prelucrări suplimentare legate de tratarea rândurilor.

Tratarea unui set de rezultate implică următoarele etape:

- **Generarea setului de rezultate prin apelarea funcției `mysql_store_result()` sau a funcției `mysql_use_result()`.** Aceste funcții returnează un pointer `MYSQL_RES` în caz de reușită, respectiv `NULL` în caz de eșec. Mai târziu, vom discuta despre diferențele dintre funcțiile `mysql_store_result()` și `mysql_use_result()`, precum și condițiile în care se alege utilizarea uneia în detrimentul alteia. Pentru moment, exemplele noastre folosesc `mysql_store_result()`, care returnează imediat rezultatele de la server și le stochează la programul client.
- **Apelarea funcției `mysql_fetch_row()` pentru fiecare rând al setului de rezultate.** Această funcție returnează o valoare `MYSQL_ROW`, care este un pointer spre un tablou de șiruri care reprezintă valorile din fiecare coloană a rândului. Modul în care utilizați rândul depinde de aplicația dumneavoastră. Puteți să afișați valorile din coloane, puteți efectua calcule statistice cu aceste valori sau orice altceva. Funcția `mysql_fetch_row()` returnează `NULL` atunci când în setul de rezultate nu au mai rămas rânduri.
- **Când ați terminat lucrul cu setul de rezultate, apelați funcția `mysql_free_result()` pentru anularea alocării memoriei pe care o folosește.** Dacă omiteți această operație, aplicația dumneavoastră va provoca „scurgeri" de memorie. (În cazul aplicațiilor cu durată mare de rulare, este important mai ales să vă debarasați în mod adecvat de seturile de rezultate; în caz contrar, veți descoperi cum controlul sistemului dumneavoastră este încet-încet preluat de procese care consumă cantități tot mai mari din resursele sistemului.)

Exemplul următor prezintă modul de prelucrare a unei interogări care returnează un set de rezultate:

```
MYSQL_RES *res_set;

if (mysql_query(conn, "SHOW TABLES FROM mysql") !=0)
   print_error(conn, "mysql_query() ratat");
else
{
   res_set = mysql_store_result (conn); /* genereaza set de rezultate */
   if (res_set == NULL)
      print_error(conn, "mysql_store_result() ratat");
   else
```

Continuare

Continuare

```
      {
         /* prelucreaza setul de rezultate, apoi
         elibereaza memoria alocata acestuia */
         process_result_set (conn, res_set);
         mysql_free_result(res_set);
      }
   }
```

Aici am trișat puțin, apelând o funcție `process_result_set()` pentru manipularea fiecărui rând. Încă nu am definit funcția respectivă, deci trebuie să facem această operație. În general, funcțiile de tratare a setului de rezultate se bazează pe un ciclu care arată astfel:

```
MYSQL_ROW rand;

while ((rand = mysql_fetch_row (res_set)) != NULL)
{
   /* se foloseste intr-un fel continutul randurilor */
}
```

Valoarea `MYSQL_ROW` returnată de funcția `mysql_fetch_row()` este un pointer spre un tablou de valori, deci accesul la fiecare valoare se reduce la accesul rândului `row[i]`, unde `i` variază între `0` și numărul de coloane ale rândului, minus o unitate.

Trebuie reținute câteva caracteristici importante ale tipului de date `MYSQL_ROW`:

- `MYSQL_ROW` este un tip pointer, deci variabilele de acest tip se declară sub forma `MYSQL_ROW rand`, nu sub forma `MYSQL_ROW *rand`.
- Șirurile dintr-un tablou `MYSQL_ROW` se termină în zero. Totuși, dacă o coloană poate conține date binare, poate conține octeți zero, deci nu trebuie să tratați valoarea ca pe un șir care se termină în zero. Determinați lungimea coloanei, pentru a ști cât de lungă este valoarea din coloană.
- Valorile tuturor tipurilor de date, chiar și ale tipurilor numerice, sunt returnate sub formă de șiruri. Dacă doriți să tratați o valoare sub formă de număr, trebuie să convertiți personal șirul.
- Valorile `NULL` sunt reprezentate prin pointeri `NULL` în cadrul tabloului `MYSQL_ROW`. Cu excepția situației când ați declarat o coloană ca fiind `NOT NULL`, trebuie să verificați întotdeauna dacă valorile din coloana respectivă sunt sau nu pointeri `NULL`.

Aplicațiile dumneavoastră pot utiliza în orice mod conținutul fiecărui rând. Din motive ilustrative, să afișăm rândurile, cu valorile coloanelor separate prin tabulatori. Pentru aceasta, avem nevoie de o funcție suplimentară, `mysql_num_fields()`, din biblioteca client; această funcție ne indică numărul de valori (coloane) pe care le conține rândul.

Iată liniile de program pentru funcția `process_result_set()`:

```
void
process_result_set (MYSQL *conn, MYSQL_RES *res_set)
{
MYSQL_ROW      rand;
```



```
unsigned int  i;

   while ((rand = mysql_fetch_row (res_set)) != NULL)
   {
      for (i = 0; i < mysql_num_fields (res_set); i++)
      {
            if (i > 0)
               fputc ('\t', stdout);
            printf ("%s", row[i] != NULL ? row[i] : "NULL");
      }
      fputc ('\n', stdout);
   }
   if (mysql_errno (conn) != 0)
      print_error (conn, "mysql_fetch_row() ratat");
   else
      printf ("%lu randuri returnate\n", (unsigned long) mysql_num_rows
               (res_set));
}
```

Funcția `process_result_set()` afișează fiecare rând într-un format cu date separate prin tabulatori (și afișând valorile `NULL` sub forma cuvântului `"NULL"`), urmat de numărul rândurilor regăsite. Numărul de rânduri respectiv este disponibil prin apelarea funcției `mysql_num_rows()`. Ca și funcția `mysql_affected_rows()`, `mysql_num_rows()` returnează o valoare de tip `my_ulonglong`, deci convertiți acea valoare la tipul `unsigned long` și folosiți pentru afișarea valorii formatul `'%lu'`.

Ciclul de preluare a rândurilor este urmat de un test pentru detectarea erorilor. În cazul în care creați setul de rezultate cu `mysql_store_result()`, o valoare `NULL` returnată de funcția `mysql_fetch_row()` are întotdeauna semnificația „nu mai există rânduri". Totuși, în cazul în care creați setul de rezultate cu `mysql_use_result()`, o valoare `NULL` returnată de funcția `mysql_fetch_row()` poate avea semnificația „nu mai există rânduri", dar poate indica și apariția unei erori. Testul nu face altceva decât să permită funcției `process_rezult_set()` să detecteze erorile, indiferent de modul în care vă creați setul de rezultate.

Această versiune a funcției `process_result_set()` execută într-un mod oarecum minimal afișarea valorilor coloanelor, o abordare care își are dezavantajele sale. De exemplu, să presupunem că executați această interogare:

```
SELECT nume, prenume, oras, stat FROM presedinte
```

Veți primi următoarele date de ieșire:

```
Adams      John   Braintree     MA
Adams      John Quincy   Braintree     MA
Arthur     Chester A.    Fairfield     VT
Buchanan         James   Mercersburg   PA
Bush       George W.     Milton MA
Carter     James E. Jr.  Plains GA
Cleveland        Grover Caldwell       NJ
...
```

Am fi putut crea un aspect estetic pentru datele de ieşire prin furnizarea de informaţii precum etichetele de coloane şi prin alinierea pe verticală a valorilor. Pentru aceasta, avem nevoie de etichete şi trebuie să cunoaştem cea mai lungă valoare din fiecare coloană. Informaţiile respective sunt disponibile, dar nu ca parte a valorilor datelor din coloane; acestea fac parte din metadatele setului de rezultate (date despre date). După ce vom generaliza puţin rutina de tratare a interogărilor, în secţiunea „Utilizarea metadatelor setului de rezultate" vom scrie un program de formatare estetică a rezultatelor.

Afişarea datelor binare

Valorile din coloane care conţin date binare ce pot include octeţi zero nu vor fi afişate în mod corespunzător dacă se foloseşte specificatorul de format '%s' al funcţiei `printf()`; această funcţie aşteaptă un şir care se termină în zero şi va afişa valoarea coloanei numai până la primul octet zero. Pentru datele binare, cel mai bine este să folosiţi lungimea coloanei, astfel încât să puteţi afişa valoarea completă. Aţi putea folosi, de exemplu, funcţiile `fwrite()` sau `putc()`.

O rutină de uz general pentru tratarea interogărilor

Exemplele anterioare de tratare a interogărilor au fost scrise ştiind dacă instrucţiunea trebuie să returneze sau nu date. Acest lucru a fost posibil deoarece interogările erau codate hard în program; am folosit o instrucţiune INSERT, care nu returnează un set de rezultate, respectiv o instrucţiune SHOW TABLES, care returnează un set de rezultate.

Totuşi, nu cunoaşteţi întotdeauna care este tipul de instrucţiune pe care îl reprezintă interogarea. De exemplu, dacă executaţi o interogare pe care o citiţi de la tastatură sau dintr-un fişier, aceasta poate fi orice instrucţiune arbitrară. Nu aveţi cum să ştiţi în avans dacă să vă aşteptaţi sau nu ca instrucţiunea respectivă să returneze rânduri. Şi atunci? Evident că nu doriţi să analizaţi interogarea pentru a determina ce tip de instrucţiune reprezintă. Oricum, nu este o operaţie la fel de simplă pe cât pare. Nu este suficient să examinaţi primul cuvânt, deoarece interogarea poate începe cu un comentariu, după cum urmează:

```
/* comentariu */ SELECT ...
```

Din fericire, nu trebuie să cunoaşteţi în prealabil tipul de interogare pentru a o putea trata în mod adecvat. Interfaţa API în C pentru MySQL permite scrierea unei rutine de uz general pentru tratarea interogărilor, care prelucrează corect orice tip de instrucţiune, indiferent dacă aceasta returnează sau nu un set de interogări.

Înainte de a scrie liniile de program pentru rutina de tratare a interogărilor, să schiţăm modul de funcţionare a acesteia:

- Emite interogarea. Dacă eşuează, am terminat.
- Dacă interogarea reuşeşte, apelează funcţia `mysql_store_result()` pentru a regăsi rândurile de la server şi pentru a crea un set de rezultate.
- Dacă funcţia `mysql_store_result()` eşuează, atunci fie interogarea nu returnează un set de rezultate, fie s-a produs o eroare în timpul încercării de regăsire a setului. Puteţi face diferenţa între aceste rezultate transferând variabila de tratare a conexiunii către funcţia `mysql_field_count()` şi verificând valoarea acesteia, după cum urmează:

  - Dacă `mysql_field_count()` este diferit de zero, indică o eroare: interogarea ar fi trebuit să returneze un set de rezultate, dar nu a făcut-o. Acest lucru se poate întâmpla din diverse motive. De exemplu, poate că setul de rezultate a fost prea mare pentru spaţiul de memorie alocat sau poate că s-a produs o cădere pe segmentul de reţea dintre client şi server în timpul preluării rândurilor.

    O uşoară complicaţie a acestei proceduri este aceea că `mysql_field_count()` nu exista anterior versiunii MySQL 3.22.24. În versiunile anterioare, se foloseşte în schimb funcţia `mysql_num_fields()`. Pentru a scrie programe care funcţionează în orice versiune a sistemului MySQL, includeţi următorul fragment de program în orice fişier care apelează funcţia `mysql_field_count()`:

    ```
    #if !defined(MYSQL_VERSION_ID) || MYSQL_VERSION_ID<32224
    #define mysql_field_count mysql_num_fields
    #endif
    ```

    Acest program impune tratarea tuturor apelurilor la funcţia `mysql_field_count()` ca apeluri la funcţia `mysql_num_fields()` pentru versiuni MySQL anterioare versiunii 3.22.24.

  - Dacă funcţia `mysql_field_count()` returnează 0, înseamnă că interogarea nu a returnat nici un set de rezultate. (Acest fapt arată că interogarea a fost o instrucţiune de tip INSERT, DELETE sau UPDATE.)

- Dacă apelul la funcţia `mysql_store_result()` reuşeşte, interogarea a returnat un set de rezultate. Prelucraţi rândurile apelând la funcţia `mysql_fetch_row()` până când returnează NULL.

Listingul următor prezintă o funcţie care prelucrează orice interogare, date fiind o variabilă de tratare a conexiunii şi un şir de interogare terminat în zero:

```
#if !defined(MYSQL_VERSION_ID) || MYSQL_VERSION_ID<32224
#define mysql_field_count mysql_num_fields
#endif

void
process_query (MYSQL *conn, char *query)
{
MYSQL_RES *res_set;
unsigned int field_count;

    if (mysql_query (conn, query) != 0) /* interogare ratata */
    {
        print_error (conn, "process_query() ratat");
        return;
    }
```

Continuare

Continuare

```
    /* interogarea a reusit; determina daca returneaza date sau nu */

    res_set = mysql_store_result (conn);
    if (res_set == NULL)                /* nu a fost returnat nici un set
                                           de rezultate */
    {
      /*
       * lipsa unui set de rezultate inseamna ca s-a produs o eroare
       * sau ca nu a fost returnat nici un set de rezultate?
       */
      if (mysql_field_count (conn) > 0)
      {
          /*
           * se astepta un set de rezultate, dar mysql_store_result() nu
           * a returnat nimic; aceasta inseamna ca s-a produs o eroare
           */
          print_error (conn, "Problema la prelucrarea setului de rezultate");
      }
      else
      {
          /*
           * nu a fost returnat nici un set de rezultate; interogarea
           * nu a returnat date (nu a fost de tip SELECT, SHOW,
           * DESCRIBE sau EXPLAIN), deci raporteaza numarul de
           * randuri afectate de interogare
           */
          printf ("%lu randuri afectate\n",
            (unsigned long) mysql_affected_rows (conn));
      }
    }
    else    /* a fost returnat un set de rezultate */
    {
      /* prelucreaza randurile, apoi elibereaza memoria alocata setului
         de rezultate */
      process_result_set (conn, res_set);
      mysql_free_result (res_set);
    }
}
```


Metode alternative pentru prelucrarea interogărilor

Versiunea funcției `process_query()` prezentată anterior are trei proprietăți:

- Folosește funcția `mysql_query()` pentru a emite interogarea.
- Folosește funcția `mysql_store_query()` pentru a regăsi setul de rezultate.
- Dacă nu se obține nici un set de rezultate, folosește `mysql_field_count()` pentru a face diferența între apariția unei erori și un set de rezultate neașteptat.

Sunt posibile abordări alternative pentru toate aceste trei aspecte ale tratării interogărilor:

- Puteți folosi un șir de interogare numerotat și funcția `mysql_real_query()` în locul unui șir de interogare care se termină în zero și al funcției `mysql_query()`.
- Puteți crea setul de rezultate apelând funcția `mysql_use_result()` în locul funcției `mysql_store_result()`.
- Puteți apela funcția `mysql_error()` în locul funcției `mysql_field_count()`, pentru a afla dacă regăsirea setului de rezultate a eșuat sau dacă nu există nici un set de regăsit.

Oricare din aceste metode (sau chiar toate) se pot folosi în cadrul funcției `process_query()`. Iată o funcție `process_real_query()` care este analogă cu `process_query()`, dar care folosește toate cele trei alternative:

```
void
process_real_query (MYSQL *conn, char *query, unsigned int len)
{
MYSQL_RES *res_set;
unsigned int field_count;

    if (mysql_real_query (conn, query, len) != 0) /* interogarea a esuat */
    {
      print_error (conn, "process_real_query() ratata");
      return;
    }

    /* interogarea a reusit; determina daca returneaza date sau nu */

    res_set = mysql_use_result (conn);
    if (res_set == NULL)          /* nu a fost returnat nici un set de
                                     rezultate */
    {
      /*
       * lipsa unui set de rezultate inseamna ca s-a produs o eroare
       * sau ca nu a fost returnat nici un set de rezultate?
       */
      if (mysql_errno (conn) != 0) /* eroare */
```

Continuare

Continuare

```
        print_error (conn, "Problema la prelucrarea setului de rezultate");
      else
      {
          /*
           * nu a fost returnat nici un set de rezultate; interogarea
           * nu a returnat date (nu a fost de tip SELECT, SHOW,
           * DESCRIBE sau EXPLAIN), deci raporteaza numarul de
           * randuri afectate de interogare
           */
          printf ("%lu randuri afectate\n",
            (unsigned long) mysql_affected_rows (conn));
      }
    }
    else /* a fost returnat un set de rezultate */
    {
      /* prelucreaza randurile, apoi elibereaza memoria alocata setului
         de rezultate */
      process_result_set (conn, res_set);
      mysql_free_result (res_set);
    }
}
```

O comparație între funcțiile mysql_store_result() și mysql_use_result()

Funcțiile mysql_store_result() și mysql_use_result() sunt asemănătoare, în sensul că ambele preiau un argument de tip variabilă de tratare a conexiunii și returnează un set de rezultate. Totuși, diferențele dintre cele două funcții sunt foarte mari. Principala diferență rezidă în modalitatea în care sunt regăsite de la server rândurile din setul de rezultate. Când o apelați, mysql_store_result() regăsește imediat toate rândurile. Funcția mysql_use_result() inițiază operația de regăsire, dar nu obține practic nici un rând. În schimb, presupune că veți apela funcția mysql_fetch_row() ulterior pentru regăsirea înregistrărilor. Aceste abordări diferite în ceea ce privește maniera de regăsire a rândurilor dau naștere la toate celelalte diferențe dintre cele două funcții. Secțiunea de față compară funcțiile, pentru a vă permite să faceți alegerea cea mai adecvată pentru o aplicație dată.

Când mysql_store_result() regăsește un set de rezultate de la server, preia rândurile, alocă memorie pentru ele și le stochează la client. Apelurile ulterioare la mysql_fetch_row() nu vor returna niciodată o eroare, deoarece pur și simplu extrag un rând din structura de date care conține deja setul de rezultate. Când funcția mysql_fetch_row() returnează NULL, înseamnă întotdeauna că ați ajuns la sfârșitul setului de rezultate.

Prin contrast, mysql_use_result() nu regăsește rânduri. În schimb, inițiază o operație de regăsire rând cu rând, pe care trebuie să o finalizați personal prin apelarea funcției

mysql_fetch_row() pentru fiecare rând. În acest caz, deși o valoare NULL returnată de mysql_fetch_row() în mod normal continuă să însemne că s-a ajuns la sfârșitul setului de rezultate, este de asemenea posibil ca în timpul comunicației cu serverul să se fi produs o eroare. Puteți face deosebirea între cele două rezultate prin apelarea funcției mysql_errno() sau a funcției mysql_error().

mysql_store_result() prezintă cerințe de memorie și de prelucrare mai ridicate decât mysql_use_result(), deoarece întregul set de rezultate este păstrat la client. Surplusul de memorie alocată și de complexitate a structurii de date este mai ridicat și un client care regăsește seturi de rezultate mari întâmpină riscul de a epuiza memoria. Dacă intenționați să regăsiți mai multe rânduri simultan, poate doriți să folosiți în schimb funcția mysql_use_result().

Funcția mysql_use_result() are cerințe de memorie mai reduse, deoarece este necesară alocarea unui spațiu de memorie pentru manipularea numai a unui singur rând la un moment dat. Aceasta poate duce la o creștere a vitezei, deoarece nu configurați o structură de date atât de complexă pentru setul de rezultate. Pe de altă parte, mysql_use_result() solicită mai mult serverul, care trebuie să memoreze rânduri din setul de rezultate până când clientul crede de cuviință că trebuie să le regăsească pe toate. Astfel, funcția mysql_use_result() nu este o alegere bună pentru anumite tipuri de clienți:

- Clienți interactivi, care se deplasează de la un rând la altul, la cererea utilizatorului. (Nu doriți ca serverul să fie forțat să aștepte trimiterea rândului următor numai pentru că utilizatorul s-a decis să ia o pauză de cafea.)
- Clienții care execută un volum mare de prelucrări între operațiile de regăsire a două rânduri.

În ambele tipuri de situații, clientul nu reușește să regăsească rapid toate rândurile din setul de rezultate. Astfel, serverul devine ocupat și faptul poate avea un impact negativ asupra altor clienți, deoarece tabelele din care regăsiți date sunt blocate la citire pe durata interogării. Toți clienții care încearcă să actualizeze aceste tabele sau să insereze rânduri în interiorul acestora sunt blocați.

Compensarea cerințelor de memorie suplimentare determinate de funcția mysql_store_result() reprezintă anumite avantaje ale accesului la întregul set de rezultate simultan. Toate rândurile din set sunt disponibile, deci aveți acces aleator la ele: funcțiile mysql_data_seek(), mysql_row_seek() și mysql_row_tell() vă permit să obțineți accesul la rânduri în orice ordine doriți. Cu funcția mysql_use_result(), puteți obține accesul la rânduri numai în ordinea în care acestea sunt regăsite de mysql_fetch_row(). Dacă intenționați să prelucrați rânduri într-o altă ordine decât cea secvențială returnată de server, trebuie să folosiți în schimb funcția mysql_store_result(). De exemplu, dacă aveți o aplicație care permite utilizatorului să se deplaseze înainte și înapoi printre rândurile selectate de o interogare, cea mai bună soluție o reprezintă aplicația mysql_store_result().

Cu funcția mysql_store_result() puteți obține anumite tipuri de informații despre coloane care nu sunt disponibile atunci când folosiți mysql_use_result(). Numărul de rânduri al setului de rezultate este obținut prin apelarea funcției mysql_num_rows(). Dimensiunile maxime ale valorilor din fiecare coloană sunt stocate în membrul

max_width al structurii de informații privind coloanele MYSQL_FIELD. Cu mysql_use_result(), mysql_num_rows() nu returnează valoarea corectă decât atunci când ați preluat toate rândurile, iar valoarea max_width este inaccesibilă, deoarece poate fi calculată numai după ce au fost examinate datele din fiecare rând.

Deoarece mysql_use_result() execută mai puține operații decât mysql_store_result(), impune o cerință inexistentă în mysql_store_result(): clientul *trebuie* să apeleze la funcția mysql_fetch_row() pentru fiecare rând din setul de rezultate. În caz contrar, toate înregistrările rămase în set devin parte a setului de rezultate al următoarei interogări și se produce o eroare de desincronizare. Acest lucru nu se întâmplă în cazul funcției mysql_store_result() deoarece, atunci când funcția respectivă returnează, toate rândurile au fost deja preluate. De fapt, dacă folosiți mysql_store_result(), nu mai trebuie să apelați personal funcția mysql_fetch_row(). Acest fapt poate fi util pentru interogări în care vă interesează numai să obțineți un rezultat nevid, nu și conținutul rezultatului. De exemplu, pentru a afla dacă un tabel my_tbl există, puteți executa această interogare:

```
SHOW TABLES LIKE "my_tbl"
```

Dacă, după ce apelați funcția mysql_store_result(), valoarea funcției mysql_num_rows() este diferită de zero, tabelul există. Nu este necesară apelarea funcției mysql_fetch_row(). (Evident, trebuie să apelați funcția mysql_free_result.)

Dacă doriți să furnizați un maximum de flexibilitate, oferiți utilizatorilor opțiunea de a selecta oricare din cele două metode de prelucrare a setului de rezultate. mysql și mysqldump sunt două programe care execută această operație. Acestea folosesc funcția mysql_store_result() în mod prestabilit, dar comută la mysql_use_result() dacă specificați opțiunea --quick.

Utilizarea metadatelor privind setul de rezultate

Seturile de rezultate nu conțin numai valorile coloanelor aferente rândurilor de date, dar și informații despre date. Aceste informații poartă numele de metadate ale setului de rezultate și includ:

- Numărul de rânduri și de coloane din fiecare set de rezultate, disponibile prin apelarea funcțiilor mysql_num_rows(), respectiv mysql_num_fields().
- Lungimea valorii fiecărei coloane dintr-un rând, disponibilă prin apelarea funcției mysql_fetch_lengths().
- Informații despre fiecare coloană, cum ar fi numele și tipul coloanei, lățimea maximă a valorilor din fiecare coloană, precum și tabelul din care provine coloana. Aceste informații sunt stocate în structurile MYSQL_FIELD, care sunt obținute în mod caracteristic prin apelarea funcției mysql_fetch_field(). Anexa F descrie structura MYSQL_FIELD în detaliu și prezintă toate funcțiile care furnizează accesul la informațiile din coloane.

Disponibilitatea metadatelor depinde în parte de metoda de prelucrare a setului de rezultate. Așa cum s-a arătat în secțiunea anterioară, dacă doriți să folosiți valorile numărului de rânduri sau ale lungimii maxime a coloanei, trebuie să creați setul de rezultate folosind mysql_store_result(), nu mysql_use_result().

Metadatele setului de rezultate sunt utile pentru luarea unor decizii privind prelucrarea datelor din setul de rezultate:

- Informațiile privind numele și lățimea coloanei sunt utile pentru producerea unor rezultate frumos formatate, cu titluri de coloană și linii aranjate pe verticală.
- Numărul de coloane se folosește pentru a determina numărul de iterații al unui ciclu care prelucrează valorile succesive ale coloanelor pentru rândurile de date. Puteți folosi numerele de rânduri sau de coloane dacă trebuie să alocați structuri de date care depind de cunoașterea numărului de rânduri sau de coloane din setul de rezultate.
- Puteți determina tipul de date al unei coloane. Această informație vă permite să vă dați seama dacă o coloană reprezintă un număr, dacă poate conține date binare și altele.

Anterior, în secțiunea „Tratarea interogărilor care returnează date", am scris o versiune a funcției process_result_set() care afișa coloane provenite din rânduri ale setului de rezultate folosind un format de separare a datelor prin tabulatori. Acest procedeu este adecvat pentru anumite scopuri (când doriți să importați datele într-o foaie de calcul tabelar), dar nu este un format de afișare estetic pentru examinare vizuală sau pentru tipărire. Să ne reamintim că versiunea anterioară a funcției process_result_set() furniza un rezultat ca acesta:

```
Adams      John    Braintree    MA
Adams      John Quincy    Braintree    MA
Arthur     Chester A.     Fairfield    VT
Buchanan         James    Mercersburg  PA
Bush       George W.      Milton MA
Carter     James E. Jr    Plains GA
Cleveland        Grover Caldwell       NJ
...
```

Să efectuăm unele modificări ale funcției process_result_set() astfel încât să genereze date de ieșire în formă tabelară prin atribuirea unui titlu pentru fiecare coloană și „încasetarea" acesteia. Versiunea revizuită va afișa aceleași rezultate, într-un format mai simplu de examinat:

nume	prenume	oras	stat
Adams	John	Braintree	MA
Adams	John Quincy	Braintree	MA
Arthur	Chester A.	Fairfield	VT
Buchanan	James	Mercersburg	PA
Bush	George W.	Milton	MA
Carter	James E., Jr.	Plains	GA
Cleveland	Grover	Caldwell	NJ
...	...	...	...

O schiță generală a algoritmului de afișare este următoarea:

1. Se determină lățimea de afișare pentru fiecare coloană.
2. Se afișează un rând de etichete de coloană „încasetate" (delimitate prin bare verticale și precedate, respectiv urmate de rânduri formate din liniuțe).
3. Se afișează valorile din fiecare rând al setului de rezultate, cu fiecare coloană încasetată (delimitată prin bare verticale) și aliniată pe verticală. În plus, numerele sunt afișate aliniate la dreapta și se afișează cuvântul "NULL" în locul valorilor NULL.

4. La sfârșit, se afișează un număr al rândurilor regăsite.

Acest exercițiu reprezintă o bună demonstrație a utilizării metadatelor setului de rezultate. Pentru a afișa rezultatele așa cum s-a arătat mai sus, trebuie să știm mai multe lucruri despre setul de rezultate, nu numai valorile datelor incluse în rânduri.

Poate vă gândiți: „Hmm, descrierea aia arată suspect de asemănător cu modul în care `mysql` își afișează rezultatele". Da, așa este, iar dumneavoastră sunteți invitat să comparați codul sursă al programului `mysql` cu programul final al funcției `process_result_set()` revizuite. Cele două nu sunt identice, dar comparația între cele două abordări ale aceleiași probleme este instructivă.

Mai întâi, trebuie să determinăm lățimea de afișare a fiecărei coloane. Listingul următor vă arată cum să procedați. Calculele se bazează în întregime pe metadatele setului de rezultate și nu se face nici o referire la valorile din rânduri:

```
MYSQL_FIELD *field;
unsigned int i, col, len;

/* determina latimile de afisare ale coloanelor */
mysql_field_seek (res_set, 0);
for (i = 0; i < mysql_num_fields (res_set); i++)
{
   field = mysql_fetch_field (res_set);
   col_len = strlen (field->name);
   if (col_len < field->max_length)
      col_len = field->max_length;
   if (col_len < 4 && !IS NOT NULL (field->flags))
      col_len = 4; /* 4 este lungimea cuvantului "NULL" */
   field->max_length = col_len; /* reinitializarea informatiilor
                                   despre coloana */
}
```

Lățimile coloanelor sunt calculate prin parcurgerea ciclică a structurilor `MYSQL_FIELD` pentru coloanele din setul de rezultate. Ne poziționăm pe prima structură prin apelarea funcției `mysql_fetch_seek()`. Apelurile ulterioare la funcția `mysql_fetch_field()` returnează pointeri spre structurile coloanelor succesive. Lățimea unei coloane pentru afișare o reprezintă cea mai mare din trei valori, fiecare depinzând de metadatele din structura cu informații privind coloanele:

- Lungimea câmpului `field->name`, titlul coloanei.
- `field->max_length`, dimensiunea celei mai lungi valori a datelor din coloană.
- Lungimea șirului `"NULL"`, în cazul în care coloana poate conține valori `NULL`. `field->flags` arată dacă acea coloană poate conține `NULL` sau nu.

Rețineți că, după ce lățimea de afișare pentru coloană a devenit cunoscută, atribuim această valoare variabilei `max_length`, care este un membru al unei structuri pe care o obținem din biblioteca client. Acest lucru este permis sau este necesar ca datele din structura `MYSQL_FIELD` să fie considerate ca fiind numai pentru citire? În mod normal, aș opta pentru a doua variantă, dar unele dintre programele client din distribuția MySQL

modifică valoarea `max_length` într-un mod asemănător, deci voi presupune că procedeul este permis. (Dacă preferați o abordare alternativă, care nu modifică `max_length`, alocați un tablou de valori `unsigned int` și stocați lățimile calculate în tabloul respectiv.)

Calculul lățimilor de afișare ascunde o capcană. Vă mai reamintiți că `max_length` nu are nici o semnificație atunci când creați un set de rezultate folosind `mysql_use_result()`. Deoarece avem nevoie de `max_length` pentru a determina lățimea de afișare a valorilor din coloană, funcționarea corectă a algoritmului impune ca setul de rezultate să fie generat folosind `mysql_store_result()`[2].

Odată cunoscute lățimile coloanelor, suntem gata de afișare. Titlurile sunt ușor de manipulat; pentru o coloană dată, folosim pur și simplu structura cu informații despre coloană indicată prin `field` și afișăm membrul `name`, folosind lățimea calculată anterior:

```
printf (" %-*s |", field->max_length, field->name);
```

Pentru date, vom parcurge ciclic rândurile din setul de rezultate, afișând valorile coloanelor pentru rândul curent în timpul fiecărei iterații. Afișarea valorilor din coloanele unui rând este oarecum ciudată, deoarece o valoare poate fi `NULL` sau poate reprezenta un număr (caz în care îl afișăm aliniat la dreapta). Valorile coloanelor sunt afișate după cum urmează, unde `row[i]` conține valorile datelor, iar `field` indică spre informațiile despre coloane:

```
if (row[i] = = NULL)
   printf (" %-*s |", field->max_length, "NULL");
else if (IS_NUM (field->type))
   printf (" %*s |", field->max_length, row[i]);
else
   printf (" %-*s |", field->max_length, row[i]);
```

Valoarea macroinstrucțiunii `IS_NUM()` este adevărată dacă tipul coloanei indicat de `field->type` este un tip numeric, precum `INT`, `FLOAT` sau `DECIMAL`.

Liniile de program finale pentru afișarea setului de rezultate se prezintă astfel. Rețineți că, deoarece afișăm de mai multe ori linii compuse din mici liniuțe, codul pentru această operație este încapsulat în propria sa funcție, `print_dashes()`:

```
void
print_dashes (MYSQL_RES *res_set)
{
MYSQL_FIELD       *field;
unsigned int      i, j;

   mysql_field_seek (res_set, 0);
   fputc ('+', stdout);
   for (i = 0; i < mysql_num_fields (res_set); i++)
   {
      field = mysql_fetch_field (res_set);
```

Continuare

[2] Membrul length al structurii `MYSQL_FIELD` vă indică lungimea maximă pe care o pot avea valorile din coloane. Aceasta poate fi o soluție utilă dacă folosiți `mysql_use_result()` în locul funcției `mysql_store_result()`. – N.A.

Continuare

```
        for (j = 0; j < field->max_length + 2; j ++)
            fputc('-', stdout);
        fputc ('+', stdout);
    }
    fputc ('\n', stdout);
}

void
process_result_set (MYSQL *conn, MYSQL_RES *res_set)
{
MYSQL_FIELD         *field;
MYSQL_ROW           row
unsigned            int i, col, len;

    /* determina latimile de afisare ale coloanelor */
    mysql_field_seek (res_set, 0);
    for (i = 0; i < mysql_num_fields (res_set); i++)
    {
        field = mysql_fetch_field (res_set);
        col_len = strlen (field->name);
        if (col_len < field->max_length)
            col_len = field->max_length;
        if (col_len < 4 && !IS NOT NULL (field->flags))
            col_len = 4; /* 4 este lungimea cuvantului "NULL" */
        field->max_length = col_len; /* reinitializarea informatiilor
                                        despre coloana */
    }

    print_dashes (res_set)
    fputc ('|', stdout);
    mysql_field_seek (res_set, 0);
    for (i = 0; i < mysql_num_fields (res_set); i++)
    {
        field = mysql_fetch_field (res_set);
        printf (" %-*s |", field->max_length, field->name);
    }
    fputc ('\n', stdout);
    print_dashes (res_set);

    while ((row = mysql_fetch_row (res_set)) != NULL)
    {
        mysql_field_seek (res_set, 0);
        fputc ('|', stdout);
```



```
        for (i = 0; i < mysql_num_fields (res_set); i++)
        {
            field = mysql_fetch_field (res_set);
            if (row[i] = = NULL)
                printf (" %-*s |", field->max_length, "NULL");
            else if (IS_NUM (field->type))
                printf (" %*s |", field->max_length, row[i]);
            else
                printf (" %-*s |", field->max_length, row[i]);
        }
        fputc ('\n', stdout);
    }
    print_dashes (res_set);
    printf ("%lu randuri returnate\n", (unsigned long)
            mysql_num_rows (res_set));
}
```

Biblioteca client MySQL furnizează numeroase modalități de acces la structurile cu informații despre coloane. De exemplu, programul din exemplul anterior obține de mai multe ori accesul la aceste structuri, folosind cicluri având următoarea formă generală:

```
mysql_field_seek (res_set, 0);
for (i = 0; i < mysql_num_fields (res_set); i++)
{
    field = mysql_fetch_field (res_set);
    ...
}
```

Cu toate acestea, combinația dintre funcțiile `mysql_field_seek()` și `mysql_fetch_field()` este numai una din modalitățile de a obține structurile `MYSQL_FIELD`. Pentru a afla despre alte modalități de obținere a structurilor cu informații despre coloane, vezi informațiile aferente funcțiilor `mysql_fetch_fields()` și `mysql_fetch_field_direct()` din Anexa F.

Client 5 – Un program interactiv de interogare

Să adunăm o bună parte din programele create până acum și să le folosim pentru a scrie un program client interactiv simplu. Acesta vă permite să introduceți interogări, le execută folosind rutina noastră de uz general pentru prelucrarea interogărilor `process_query()` și afișează rezultatele folosind programul de formatare a datelor de ieșire `process_result_set()`, creat în secțiunea precedentă.

`client5` va fi similar din multe puncte de vedere cu `mysql`, deși nu va avea, desigur, un număr atât de mare de caracteristici. Există numeroase restricții cu privire la datele de intrare care vor fi acceptate de `client5`:

- Fiecare linie de intrare trebuie să conțină o singură interogare completă.
- Interogările nu trebuie să se încheie cu punct și virgulă sau cu `\g`.
- Comenzi precum `quit` nu sunt recunoscute; în schimb, folosiți Control-D pentru a termina execuția programului.

Se observă că redactarea programului `client5` este aproape banală (sub 10 linii de program noi). Aproape toate elementele necesare sunt furnizate de scheletul programului client (`client4.c`) și de alte programe pe care le-am scris deja. Singurul lucru care mai trebuie adăugat este un ciclu care adună datele de intrare și le execută.

Pentru a construi programul `client5`, începeți prin a copia scheletul de program client `client4.c` în programul `client5.c`. Apoi adăugați la acesta liniile de program aferente funcțiilor `process_query()`, `process_result_set()` și `print_dashes()`. În final, în programul `client5.c`, căutați linia din funcția `main()` care are următorul conținut:

```
/* aici se insereaza codul aplicatiei propriu-zise */
```

Apoi înlocuiți-o cu următorul ciclu `while`:

```
while (1)
{
char buf[1024];
   fprintf (stderr, "query> "); /* prompt de afisare */
   if (fgets (buf, sizeof (buf), stdin) = = NULL)
                                 /* citeste interogarea */
      break;
   process_query (conn, buf);    /* executa interogarea */
}
```

Compilați programul `client5.c` pentru a produce fișierul `client5.o`, legați `client5.o` cu `common.o` și cu biblioteca client pentru a genera fișierul `client5` și ați terminat! Aveți un program client MySQL interactiv, care poate executa orice interogare și afișează rezultatele.

Alte aspecte

Această secțiune tratează numeroase subiecte care nu s-au putut încadra foarte bine în evoluția de la `client1` la `client5`:

- Utilizarea datelor din setul de rezultate pentru a calcula un rezultat, după utilizarea metadatelor din setul de rezultate pentru a contribui la verificarea faptului că datele sunt adecvate pentru calculul dumneavoastră.
- Modul de tratare a datelor dificil de inserat în interogări.
- Modul de lucru cu datele de tip imagine.
- Modul de obținere a informațiilor referitoare la structura tabelelor dumneavoastră.
- Greșeli comune de proiectare în MySQL și modul de evitare a acestora.

Efectuarea de calcule cu seturile de rezultate

Până acum, ne-am concentrat asupra utilizării metadatelor aferente seturilor de rezultate mai ales pentru afișarea datelor din rânduri, dar este evident că vor exista situații când datele dumneavoastră vor trebui folosite și în alte moduri, nu numai afișate. De exemplu, puteți efectua calcule statistice în funcție de valorile datelor, folosind metadatele pentru a vă asigura că datele se conformează cerințelor pe care doriți ca acestea să le satisfacă. Ce

fel de cerințe? Începătorii vor dori probabil să verifice dacă o coloană cu al cărei conținut intenționează să efectueze calcule numerice conține cu adevărat numere!

Listingul următor prezintă o funcție simplă, denumită `summary_stats()`, care preia un set de rezultate și un index de coloană și produce statistici de sumar pentru valorile din coloană. De asemenea, funcția raportează numărul de valori care lipsesc, pe care le detectează căutând valorile `NULL`. Aceste calcule implică două cerințe pe care trebuie să le satisfacă datele, deci `summary_stats()` le verifică folosind metadatele aferente setului de rezultate:

- Coloana specificată trebuie să existe (adică indexul coloanei trebuie să se găsească în domeniul care conține numărul de coloane din setul de rezultate).
- Coloana trebuie să conțină valori numerice.

Dacă aceste condiții nu sunt valabile, `summary_stats()` pur și simplu afișează un mesaj de eroare și returnează. Liniile de program sunt următoarele:

```
void
summary_stats (MYSQL_RES *res_set, unsigned int col_num)
{
MYSQL_FIELD*camp;
MYSQL_ROW           rand;
unsigned int        n,lipsa;
double              val, suma, suma_patrate, var;

   /* verifica indeplinirea cerintelor pentru date */
   if (mysql_num_fields (res_set) < col_num)
   {
      print_error (NULL, "numar de coloana incorect");
      return;
   }
   mysql_field_seek (res_set, 0);
   field = mysql_fetch_field (res_set);
   if (!IS_NUM (field->type))
   {
      print_error (NULL, "coloana nu este numerica");
      return;
   }

   /* calculeaza statisticile de sumar */

   n = 0;
   lipsa = 0;
   suma = 0;
   suma_patrate = 0;
```

Continuare

Continuare

```
mysql_data_seek (res_set, 0);
while ((rand = mysql_fetch_row (res_set)) != NULL)
{
   if (rand[col_num] == NULL)
        lipsa++;
   else
   {
        n++;
        val = atof (rand[col_num]); /* conversie din sir in numar */
        suma += val;
        suma_patrate += val * val;
   }
}
if (n = = 0)
   printf ("Nu sunt observatii\n");
else
{
   printf ("Numar de observatii: %lu\n", n);
   printf ("Observatii lipsa: %lu\n", lipsa);
   printf ("Suma: %g\n", suma);
   printf ("Media: %g\n", suma / n);
   printf ("Suma patratelor: %g\n", suma_patrate);
   var = ((n * suma_patrate) – (suma * suma)) / (n * (n - 1));
   printf ("Dispersie: %g\n", var);
   printf ("Abatere standard": %g/n, sqrt (var));
}
}
```

Observați apelul la funcția mysql_data_seek() care precede ciclul funcției mysql_fetch_loop(). Apelul respectiv are rolul de a vă permite să apelați funcția summary_stats() de mai multe ori, pentru a obține același set de rezultate (în caz că doriți să calculați date statistice pentru mai multe coloane). La fiecare apelare a funcției summary_stats(), aceasta „dă înapoi" până la începutul setului de rezultate. (Acest fapt presupune că ați creat setul de rezultate cu mysql_store_result(). Dacă îl creați cu mysql_use_result(), puteți prelucra rândurile numai în ordine și numai o dată.)

summary_stats() este o funcție relativ simplă, dar ar trebui să vă ofere o idee cu privire la modul în care puteți programa calcule mai complexe, cum este regresia celor mai mici pătrate pe două coloane sau statistici standard, cum este un test *t*.

Codificarea în interogări a datelor problematice

Valorile de date care conțin zerouri, ghilimele sau backslash-uri, dacă sunt inserate literal într-o înregistrare, pot cauza probleme atunci când încercați să executați interogarea. Expunerea următoare prezintă natura dificultății și modul de rezolvare a acesteia.

Să presupunem că doriți să construiți o interogare SELECT în funcție de conținutul șirului încheiat prin zero și indicat prin nume:

```
char query[1024];
sprintf (query, "SELECT * FROM tabel WHERE nume='%s'", nume);
```

Dacă valoarea lui nume este ceva de genul "O'Malley, Brian", interogarea rezultată este incorectă, deoarece în cadrul unui șir delimitat prin ghilimele apar alte ghilimele:

```
SELECT * FROM tabel WHERE nume='O'Malley, Brian'
```

Trebuie să tratați ghilimelele într-un mod aparte, astfel încât serverul să nu le interpreteze ca fiind sfârșitul numelui. O modalitate constă în dublarea ghilimelelor din interiorul șirului. Aceasta este convenția ANSI pentru SQL. MySQL înțelege această convenție și mai permite ca ghilimelele să fie precedate de un backslash:

```
SELECT * FROM tabel WHERE nume='O''Malley, Brian'
SELECT * FROM tabel WHERE nume='O\'Malley, Brian'
```

O altă situație problematică implică utilizarea într-o interogare a unor date binare arbitrare. Acest lucru se întâmplă, de exemplu, în aplicații care stochează imagini într-o bază de date. Deoarece o valoare binară poate conține orice caracter, nu poate fi inclusă ca atare într-o interogare, în condiții de siguranță.

Pentru a rezolva această problemă, folosiți mysql_escape_string(), care codifică acele caractere speciale, pentru a permite utilizarea lor în șirurile delimitate prin ghilimele. Caracterele pe care funcția mysql_escape_string() le consideră speciale sunt caracterul nul, ghilimelele simple și duble, backslash-ul, linia nouă, returul de car și Control-Z. (Ultimul apare în contexte Windows.)

Când trebuie să folosiți mysql_escape_string()? Răspunsul cel mai sigur este „întotdeauna". Totuși, dacă sunteți sigur de forma datelor dumneavoastră și știți că totul este în ordine – poate deoarece ați efectuat anterior verificări pentru validare – nu trebuie să le codificați. De exemplu, dacă lucrați cu șiruri despre care știți că reprezintă numere de telefon corecte, alcătuite integral din cifre și liniuțe, nu trebuie să apelați mysql_escape_string(). În caz contrar, probabil că trebuie să apelați funcția respectivă.

mysql_escape_string() codifică acele caractere problematice transformându-le în secvențe de două caractere care încep cu un backslash. De exemplu, un octet nul devine \0, unde 0 este un caracter 0 ASCII care poate fi afișat, nu un caracter nul. Caracterele backslash, ghilimelele simple și duble se transformă în \\, \' și \".

Pentru a folosi funcția mysql_escape_string(), invocați-o astfel:

```
to_len = mysql_escape_string (to_str, from_str, from_len);
```

mysql_escape_string() codifică șirul from_str și scrie rezultatul în to_str. De asemenea, adaugă un zero de final, care este convenabil deoarece puteți folosi șirul rezultant cu funcții precum strcpy() și strlen().

from_str indică spre un buffer de tip char, care conține șirul ce trebuie codificat. Acest șir poate conține orice, inclusiv date binare. to_str indică spre un buffer char existent, unde doriți să fie scris șirul codificat; *nu transferați* un pointer neinițializat sau NULL, așteptând ca funcția mysql_escape_string() să aloce spațiu în mod automat. Lungimea bufferului indicat de to_str trebuie să fie de cel puțin (from_len*2)+1 octeți. (Este posibil ca fiecare caracter din from_str să necesite codificarea cu două caractere; octetul suplimentar este destinat zeroului de final.)

from_len şi to_len sunt valori de tipul unsigned int. from_len indică lungimea datelor din from_str; este necesară furnizarea lungimii, deoarece from_str poate conţine octeţi nuli şi nu poate fi tratat drept un şir terminat în zero. to_len, valoarea returnată de funcţia mysql_escape_string(), este lungimea efectivă a şirului codificat rezultant, fără a număra zeroul final.

Când funcţia mysql_escape_string() returnează, rezultatul codificat în to_str poate fi tratat ca un şir care se termină în zero, deoarece toate zerourile din from_str sunt codificate sub forma secvenţei afişabile \0.

Pentru a rescrie codul de construcţie al instrucţiunii SELECT astfel încât să funcţioneze chiar şi pentru nume care conţin ghilimele, vom proceda astfel:

```
char query[1024], *p;

p = strcpy (query, "SELECT * FROM tabel WHERE nume='");
p += strlen (p);
p += mysql_escape_string (p, nume, strlen (nume));
p = strcpy (p, "'");
```

Da, e urât. Dacă doriţi să simplificaţi puţin, cu preţul utilizării unui al doilea buffer, procedaţi astfel:

```
char query[1024], buf[1024];

(void) mysql_escape_string (buf, nume, strlen (nume));
sprintf (query, "SELECT * FROM tabel WHERE nume='%s'", buf);
```

Lucrul cu date sub formă de imagini

Una dintre sarcinile pentru care funcţia mysql_escape_string() este esenţială implică încărcarea datelor sub formă de imagine într-un tabel. Această secţiune vă arată cum să procedaţi. (Expunerea de faţă este valabilă şi pentru orice formă de date binare.)

Să presupunem că doriţi să citiţi imagini din fişiere şi să le stocaţi într-un tabel, alături de un identificator unic. Tipul BLOB reprezintă o opţiune bună pentru datele binare, deci puteţi folosi o specificaţie de tabel ca aceasta:

```
CREATE TABLE imagini
{
    imagine_id INT NOT NULL PRIMARY KEY,
    imagine_data BLOB
}
```

Pentru a vă procura efectiv o imagine dintr-un fişier situat în tabelul imagini, funcţia următoare, incarca_imagine(), execută această operaţie, date fiind un număr de identificare şi un pointer spre un fişier deschis care conţine datele în format imagine:

```
int
incarca_imagine (MYSQL *conn, int id, FILE *f)
{
char            query[1024*100], buf[1024*10], *p;
unsigned int    from_len;
```



```
int             status;

    sprintf (query, "INSERT INTO imagini VALUES (%d,'", id);
    p = query + strlen (query);
    while ((from_len = fread (buf, 1, sizeof (buf), f)) > 0)
    {
        /* nu depasi finalul bufferului de interogare! */
        if (p + (2*from_len) + 3 > query + sizeof (query))
        {
            print_error (NULL, "imagine prea mare");
            return (1);
        }
        p += mysql_escape_string (p, buf, from_len);
    }
    (void) strcpy (p, "')");
    status = mysql_query (conn, query);
    return (status);
}
```

incarca_imagine() nu alocă un buffer de interogare foarte mare (100K), deci funcţionează numai pentru imagini relativ simple. Într-o aplicaţie din lumea reală, puteţi aloca bufferul în mod dinamic, în funcţie de dimensiunea fişierului imagine.

Manipularea datelor sub formă de imagine (sau a oricăror date binare) pe care le obţineţi dintr-o bază de date nu constituie o problemă la fel de mare ca inserţia lor pentru a putea fi utilizate, deoarece valorile de date sunt disponibile în formă brută în variabila MYSQL_ROW, iar lungimile sunt disponibile prin apelarea funcţiei mysql_fetch_lengths(). Pur şi simplu nu uitaţi să trataţi valorile ca şiruri numărate, nu ca şiruri care se încheie cu zero.

Obţinerea informaţiilor referitoare la tabele

MySQL vă permite să obţineţi informaţii despre structura tabelelor dumneavoastră folosind oricare dintre aceste interogări (care sunt echivalente):

```
DESCRIBE nume_tabel
SHOW FIELDS FROM nume_tabel
```

Ambele instrucţiuni seamănă cu SELECT, în sensul că returnează un set de rezultate. Pentru a afla date despre coloanele din tabel, tot ce trebuie să faceţi este să prelucraţi rândurile din rezultat pentru a extrage informaţiile de care aveţi nevoie. De exemplu, dacă emiteţi o instrucţiune DESCRIBE imagini din clientul mysql, acesta va returna următoarea informaţie:

Field	Type	Null	Key	Default	Extra
imagine_id	int(11)		PRI	0	
imagine_data	blob	YES		NULL	

Dacă executaţi aceeaşi interogare din propriul dumneavoastră client, veţi primi aceleaşi informaţii (fără casete).

Dacă doriți informații numai despre o singură coloană, folosiți această interogare:

```
SHOW FIELDS FROM nume_tabel LIKE "nume_coloana"
```

Interogarea va returna aceleași coloane, dar un singur rând (sau nici un rând, în cazul în care coloana nu există).

Greșeli de programare a programelor client care trebuie evitate

Această secțiune discută despre unele erori comune de programare a interfețelor API pentru MySQL scrise în C, precum și despre modul de evitare a acestora. (Aceste probleme apar periodic în lista de corespondență despre MySQL; nu le-am inventat eu.)

Greșeala 1: Utilizarea unor pointeri neinițializați ai variabilelor de tratare a conexiunilor

În exemplele prezentate în acest capitol, am apelat funcția `mysql_init()` transferându-i acesteia un argument `NULL`. Astfel, se indică funcției `mysql_init()` să aloce și să inițializeze o structură `MYSQL` și să returneze un pointer spre aceasta. O altă abordare o constituie transferul unui pointer spre o structură `MYSQL`. În acest caz, `mysql_init()` va inițializa acea structură și va returna un pointer către aceasta fără a aloca structura însăși. Dacă doriți să folosiți această a doua metodă, aceasta poate duce la anumite probleme dificil de depistat. Următoarea expunere va scoate în evidență unele probleme de evitat.

Dacă transferați un pointer funcției `mysql_init()`, acesta trebuie să indice spre ceva... Să luăm această componentă de program:

```
main()
{
MYSQL     *conn;
   mysql_init (conn);
   ...
}
```

Problema este că `mysql_init()` primește un pointer, dar pointerul nu indică spre nimic semnificativ. `conn` este o variabilă locală și, implicit, constituie un spațiu de stocare neinițializat, care poate indica oriunde atunci când funcția `main()` își începe execuția. Aceasta înseamnă că `mysql_init()` va folosi pointerul și va scrie undeva într-o zonă aleatoare de memorie. Dacă aveți noroc, `conn` va indica undeva în afara spațiului de adrese al programului dumneavoastră și sistemul va termina imediat execuția programului, deci veți sesiza apariția problemei încă de la primele linii de program. Dacă nu sunteți atât de norocos, `conn` va indica spre date pe care le folosiți mai târziu în program, iar dumneavoastră nu veți sesiza apariția unei probleme decât atunci când programul încearcă efectiv să folosească datele respective. În acest caz, problema va părea că se produce într-o fază a programului mult mai avansată decât aceea din care provine efectiv și poate fi mult mai dificil de depistat.

Iată un program „cu probleme" asemănător:

```
MYSQL *conn;

main()
```



```
{
   mysql_init(conn);
   mysql_real_connect (conn, ...)
   mysql_query (conn, "SHOW DATABASES");
   ...
}
```

În acest caz, `conn` este o variabilă globală, deci este inițializată la `0` (adică `NULL`) înainte de pornirea programului. `mysql_init()` vede un argument `NULL`, deci se inițializează și alocă o nouă variabilă de tratare a conexiunii. Din păcate, `conn` este încă `NULL`, deoarece niciodată nu îi este atribuită vreo valoare. De îndată ce transferați `conn` unei funcții din interfața API în C pentru MySQL care impune o variabilă de tratare a conexiunii diferită de `NULL`, programul dumneavoastră va cădea. Pentru ambele programe, remediul îl constituie asigurarea faptului că variabila `conn` are o anumită valoare. De exemplu, o puteți inițializa la adresa unei structuri `MYSQL` deja alocate:

```
MYSQL conn_struct, *conn = &conn_struct;
...
mysql_init (conn);
```

Totuși, soluția recomandată (și mai ușoară!) este pur și simplu de a transfera `NULL` în mod explicit funcției `mysql_init()`, de a permite acelei funcții să aloce automat structura `MYSQL` și să atribuiți variabilei `conn` valoarea returnată:

```
MYSQL *conn;
...
conn = mysql_init (NULL);
```

În orice caz, nu uitați să testați valoarea returnată a funcției `mysql_init()`, pentru a vă asigura că nu este `NULL`.

Greșeala 2: Lipsa testului validității unui set de rezultate

Nu uitați să verificați starea apelurilor de unde vă așteptați să obțineți un set de rezultate. Programul de mai jos nu face asta:

```
MYSQL_RES *res_set;
MYSQL_ROW row;

res_set = mysql_store_result (conn);
while ((row = mysql_fetch_row (res_set)) != NULL)
{
   /* prelucrare rand */
}
```

Din păcate, dacă `mysql_store_result()` eșuează, `res_set` este `NULL` și ciclul `while` nici măcar nu mai trebuie executat. Testați valoarea returnată de funcțiile care returnează seturi de rezultate, pentru a vă asigura că aveți efectiv un „material de lucru".

Greșeala 3: Ignorarea valorilor NULL din coloane

Nu uitați să verificați dacă valorile coloanelor din tabloul `MYSQL_ROW` returnat de către funcția `mysql_fetch_row()` sunt pointeri `NULL`. Pe unele calculatoare, programul următor „cade" dacă `row[i]` este `NULL`:

```
for (i = 0; i < mysql_num_fields (res_set); i++)
{
   if (i > 0)
      fputc ('\t', stdout);
   printf ("%s", row[i]);
}
fputc ('\n', stdout);
```

Partea cea mai gravă a acestei greşeli este că unele versiuni ale funcţiei `printf()` sunt „iertătoare“ şi afişează `"(null)"` pentru pointeri `NULL`, ceea ce vă permite să scăpaţi fără a remedia problema. Dacă oferiţi programul dumneavoastră unui prieten care dispune de o funcţie `printf()` mai puţin mărinimoasă, programul cade şi prietenul dumneavoastră ajunge la concluzia că sunteţi un programator de doi bani. Ciclul de mai sus ar fi trebuit scris astfel:

```
for (i = 0; i < mysql_num_fields (res_set); i++)
{
   if (i > 0)
      fputc ('\t', stdout);
   printf ("%s", row[i] != NULL ? row[i] : "NULL");
}
fputc ('\n', stdout);
```

Singura dată când nu trebuie să verificaţi dacă valoarea dintr-o coloană este `NULL` este atunci când aţi determinat deja, din structura de informaţii privind coloana, că funcţia `IS_NOT_NULL()` are valoarea „adevărat“.

Greşeala 4: A transfera buffere de rezultat lipsite de sens

Funcţiile din biblioteca client care se aşteaptă ca dumneavoastră să furnizaţi buffere, în general pretind ca aceste buffere să şi existe. Programul de mai jos încalcă acest principiu:

```
char *from_str = "un sir";
char *to_str;
unsigned int len;

len = mysql_escape_string (to_str, from_str, strlen (from_str));
```

Care este problema? `to_str` trebuie să indice spre un buffer existent. În acest exemplu, nu o face; indică spre o locaţie aleatoare. Nu transferaţi un pointer neiniţializat ca argument `to_str` al funcţiei `mysql_escape_string()` decât dacă doriţi ca aceasta să se împiedice cu entuziasm peste cine ştie ce zonă de memorie aleatoare.

CAPITOLUL 7

Interfaţa API pentru Perl DBI

Capitolul de faţă prezintă modul de utilizare a interfeţei Perl DBI pentru MySQL. Nu este discutată filozofia sau arhitectura DBI. Pentru informaţii privind aceste aspecte ale DBI (mai ales în comparaţie cu interfeţele API în C şi PHP), consultaţi capitolul 5, „Introducere în programarea MySQL“.

Exemplele din acest capitol pornesc de la baza noastră de date demonstrativă, şi anume `samp_db`, folosind tabelele necesare pentru proiectul de evidenţă a rezultatelor şcolare, respectiv pentru Liga istorică. Pentru a utiliza la maximum acest capitol, trebuie să ştiţi unele lucruri despre Perl. Dacă nu, vă puteţi scrie propriile dumneavoastră scripturi prin simpla copiere a programului demonstrativ prezentat aici. Dar probabil că o carte bună de Perl va constitui o investiţie solidă pentru dumneavoastră. O asemenea carte este *Programming Perl, Second Edition*, de Wall, Christiansen, Schwartz şi Potter (O'Reilly, 1996).

În prezent, DBI se află la versiunea 1.13, deşi majoritatea celor prezentate aici se aplică şi versiunilor anterioare 1.xx. Caracteristicile descrise în capitolul de faţă şi care nu există în versiunile anterioare au fost menţionate în acest sens.

DBI pentru MySQL necesită o versiune de Perl cel puţin la fel de recentă ca versiunea 5.004_05. De asemenea, trebuie să instalaţi `Msql-Mysql-modules` şi modulele Perl Data-Dumper, precum şi fişierele antet şi biblioteca client C pentru MySQL. Dacă intenţionaţi să scrieţi scripturi DBI bazate pe Perl, probabil că veţi dori să folosiţi modulul CGI.pm. În acest capitol, modulul respectiv este folosit în conjuncţie cu serverul Web Apache. Dacă trebuie să vă procuraţi oricare dintre aceste pachete, vezi Anexa A, „Obţinerea şi instalarea programelor“. Instrucţiunile pentru procurarea scripturilor cu caracter de exemplu prezentate în acest capitol sunt de asemenea date în anexa respectivă. Puteţi descărca scripturile, pentru a evita să le tastaţi personal.

În majoritatea situaţiilor, acest capitol descrie metodele şi variabilele Perl DBI numai în măsura în care acestea sunt necesare pentru expunerea de faţă. Pentru o listă comprehensivă a tuturor metodelor şi variabilelor, vezi Anexa G, „Referinţă API Perl DBI“. Puteţi folosi anexa respectivă ca referinţă pentru alte informaţii privind orice componentă DBI pe care încercaţi să o folosiţi. Prin rularea următoarelor comenzi se poate obţine accesul la documentaţia electronică:

```
% perldoc DBI
% perldoc DBI::FAQ
% perldoc DBD::mysql
```

La nivelul driverului de baze de date (DBD), driverul pentru MySQL este construit peste biblioteca client C pentru MySQL și ca atare împrumută unele dintre caracteristicile acesteia. Vezi capitolul 6, „Interfața API MySQL pentru C", pentru mai multe informații despre biblioteca respectivă.

Caracteristici ale scripturilor Perl

Scripturile Perl nu sunt altceva decât fișiere text, iar dumneavoastră le puteți crea folosind orice editor de texte. Toate scripturile Perl din acest capitol respectă convenția UNIX de a utiliza o primă linie care începe cu `#!`, urmată de numele de cale al programului folosit pentru executarea scriptului. Linia pe care o utilizez eu este următoarea:

```
#! /usr/bin/perl
```

Va trebui să modificați linia `#!` dacă numele căii de acces spre Perl este diferit în sistemul dumneavoastră, de exemplu `/usr/local/bin/perl5` sau `/opt/bin/Perl`. În caz contrar, scripturile Perl nu vor rula corect în sistemul dumneavoastră.

Am inclus un spațiu după secvența `#!`, deoarece unele sisteme interpretează secvența `#! /` ca pe un număr magic din 4 octeți, ignoră linia dacă spațiul lipsește și astfel tratează scriptul ca pe un script de interpretor.

Sub UNIX, un script Perl trebuie să fie executabil, pentru a putea fi rulat prin simpla tastare a numelui său. Pentru a face un fișier script executabil, modificați-i modul fișier după cum urmează:

```
% chmod +x nume_script
```

Dacă folosiți ActiveState Perl sub Windows, nu faceți scripturile executabile. În schimb, rulați un script ca acesta:

```
C:\> perl nume_script
```

Elemente fundamentale ale modulului Perl DBI

Această secțiune furnizează informații elementare despre Perl DBI, în speță informațiile de care aveți nevoie pentru a vă scrie propriile scripturi și pentru a înțelege scripturile scrise de alții. Dacă sunteți un cunoscător în materie de DBI, puteți trece direct la secțiunea „Utilizarea DBI".

Tipuri de date DBI

Din anumite puncte de vedere, utilizarea interfeței API pentru Perl DBI este similară cu utilizarea bibliotecii client C descrisă în capitolul 6. Când folosiți biblioteca client C, apelați funcții și obțineți acces la date legate de MySQL mai ales prin intermediul pointerilor către structuri sau către tablouri. Când folosiți interfața API pentru DBI, continuați să apelați funcții și să folosiți pointeri la structuri, cu excepția faptului că funcțiile se numesc *metode*, pointerii se numesc *referințe*, variabilele pointer se numesc *variabile de manipulare*, iar structurile spre care indică variabilele de manipulare se numesc *obiecte*.

DBI folosește numeroase categorii de variabile de manipulare. În documentația DBI, acestea au tendința de a fi specificate prin numele convenționale prezentate în tabelul 7.1.

Modalitatea în care folosiți aceste variabile va deveni evidentă pe parcurs. De asemenea, sunt folosite numeroase nume convenționale pentru variabile, altele decât variabilele de manipulare (vezi tabelul 7.2). Nu vom folosi toate aceste nume de variabile în capitolul de față, dar este util să le cunoașteți atunci când citiți scripturi DBI scrise de alte persoane.

Tabelul 7.1 Nume convenționale ale variabilelor de manipulare folosite în Perl DBI

Nume	Semnificație
`$dbh`	Variabilă de manipulare a unui obiect bază de date
`$sth`	Variabilă de manipulare a unui obiect instrucțiune (interogare)
`$fh`	Variabilă de manipulare a unui fișier deschis
`$h`	Variabilă de manipulare „generală"; semnificația depinde de conținut

Tabelul 7.2 Nume convenționale ale variabilelor, altele decât variabilele de manipulare folosite în Perl DBI

Nume	Semnificație
`$rc`	Codul returnat de operații care returnează „adevărat" sau „fals"
`$rv`	Valoarea returnată de operații care returnează un întreg
`$rows`	Valoarea returnată de operații care returnează un număr de rânduri
`@ary`	Un tablou (listă) care reprezintă un rând de valori returnate de o interogare

Un script DBI simplu

Să începem de la un script simplu, `dump_members`, care ilustrează numeroase concepte standard ale programării DBI, cum ar fi conectarea și deconectarea de la serverul MySQL, emiterea interogărilor și regăsirea datelor. Acest script produce drept rezultat lista membrilor Ligii istorice, într-un format cu datele delimitate prin tabulatori. Formatul nu este interesant în sine; în acest moment, este mai important să vedem cum se folosește DBI decât să generăm date de ieșire cu un aspect estetic.

Scriptul `dump_members` se prezintă astfel:

```
#! /usr/bin/perl

# dump_members - afiseaza lista membrilor Ligii istorice

use DBI;
use strict;

my ($dsn) = "DBI:mysql:samp_db:localhost"; # nume sursa date
my ($user_name) = "paul";      # numele utilizatorului
my ($password) = "secret"      # parola
my ($dbh, $sth);               # variabile de manipulare pentru baza de
                               date si instructiune
```

Continuare

Continuare

```
my (@ary);                      # tablou pentru randurile returnate de
                                  interogare

# conectarea la baza de date
$dbh = DBI->connect ($dsn, $user_name, $password, { RaiseError => 1 });

# emiterea interogarii
$sth = $dbh->prepare ("SELECT nume, prenume, sufix, email,"
   . "strada, oras, stat, cod_postal, telefon FROM membru
   . "ORDER BY nume");
$sth->execute ();

# citeste rezultatele interogarii, apoi face curatenie
while (@ary = $sth->fetchrow_array ())
{
   print join ("\t", @ary), "\n";
}
$sth->finish ();

$dbh->disconnect ();
exit (0);
```

Pentru a testa personal scriptul, fie îl descărcaţi (vezi Anexa A), fie îl creaţi folosind un editor de texte şi apoi îl faceţi executabil pentru a-l putea rula. Probabil că va trebui să modificaţi măcar o parte din parametrii de conexiune (numele gazdei, numele bazei de date, numele de utilizator, parola). Acest fapt este valabil şi pentru celelalte scripturi DBI din capitolul de faţă. În mod prestabilit, permisiunile pentru scripturile din acest capitol care pot fi descărcate din Internet sunt astfel configurate încât scripturile pot fi citite numai de dumneavoastră. Vă propun să nu modificaţi această proprietate, dacă vă inseraţi propriul nume de utilizator MySQL şi propria parolă, pentru ca aceste valori să nu poată fi citite de alte persoane. Mai târziu, în secţiunea „Specificarea parametrilor de conexiune", vom vedea cum se pot obţine parametrii dintr-un fişier cu opţiuni, în loc de a-i insera direct în script.

Acum, să parcurgem scriptul instrucţiune cu instrucţiune. Prima linie conţine indicatorul standard de localizare a limbajului Perl:

```
#! /usr/bin/perl
```

Această linie face parte din toate scripturile pe care le vom discuta în capitolul de faţă; nu vom mai vorbi despre ea în viitor.

Este o idee bună să includeţi în script cel puţin o descriere minimală a scopului său, deci linia următoare este un comentariu care furnizează oricărei persoane care examinează scriptul o idee asupra finalităţii acestuia:

```
# dump_members - afiseaza lista membrilor Ligii istorice
```

Textul cuprins între caracterul # şi până la sfârşitul liniei se consideră comentariu. Merită să inseraţi comentarii în scripturi, comentarii care explică modul de funcţionare a acestora.

În continuare, avem două linii use:

```
use DBI;
use strict;
```

`use DBI` comunică interpretorului Perl că trebuie să folosească modulul DBI. În lipsa acestei linii, se va produce o eroare de îndată ce încercaţi să întreprindeţi orice acţiune legată de DBI. Totuşi, nu trebuie să indicaţi modulul de nivel DBD dorit. DBI activează automat modulul corect atunci când vă conectaţi la baza dumneavoastră de date.

`use strict` indică programului Perl să vă oblige să declaraţi variabilele înainte de a le folosi. Puteţi scrie scripturi fără a insera o linie `use strict`, dar aceasta este utilă pentru detectarea greşelilor, deci vă recomand să o folosiţi întotdeauna. De exemplu, când modul strict este activat, dacă declaraţi o variabilă `$my_var`, dar apoi faceţi referire la variabila respectivă în mod eronat, sub numele `$mv_var`, veţi primi următorul mesaj atunci când rulaţi scriptul:

```
Global symbol "$mv_var" requires explicit package name at line n
```

(Simbolul global „$mv_var" impune un nume explicit de pachet la linia *n*)

Când vedeţi aşa ceva, vă spuneţi: „Ce? `$mv_var`? N-am folosit niciodată o variabilă cu numele acela!" Apoi vă examinaţi linia n din script, depistaţi problema şi o remediaţi. În lipsa modului strict, Perl nu va ridica obiecţii în ceea ce priveşte `$mv_var`; pur şi simplu creează o variabilă cu numele respectiv şi de valoare `undef` (de la `undefined` - nedefinit), o foloseşte fără să facă nazuri, iar dumneavoastră vă întrebaţi de ce oare nu merge scriptul.

Deoarece lucrăm în modul strict, vom declara variabilele pe care le foloseşte scriptul:

```
my ($dsn) = "DBI:mysql:samp_db:localhost";    # nume sursa date
my ($user_name) = "paul";                     # numele utilizatorului
my ($password) = "secret"                     # parola
my ($dbh, $sth);                              # variabile de manipulare
                                                pentru baza de date si
                                                instructiune
my (@ary);                                    # tablou pentru randurile
                                                returnate de interogare
```

Acum, suntem pregătiţi să ne conectăm la baza de date:

```
# conectarea la baza de date
$dbh = DBI->connect ($dsn, $user_name, $password, { RaiseError => 1 });
```

Apelul la funcţia `connect()` este invocat sub forma `DBI->connect()`, deoarece această funcţie este o metodă a clasei `DBI`. Nu trebuie să ştiţi exact ce înseamnă asta; nu este nimic altceva decât o mică mostră de jargon orientat spre obiecte, aşa, ca să vă doară capul. (Dacă vreţi să ştiţi cu tot dinadinsul, înseamnă că funcţia `connect()` este o funcţie care „aparţine" modulului DBI.) Funcţia `connect()` preia mai multe argumente:

- **Sursa de date.** (Denumită deseori numele sursei de date sau DSN – abreviere de la *data source name*.) Formatele surselor de date sunt determinate de cerinţele modulului DBD pe care doriţi să-l folosiţi. Pentru driverul MySQL, formatele permise includ oricare din următoarele variante:

```
"DBI:mysql:nume_baza_de_date"
"DBI:mysql:nume_baza_de_date:nume_gazda"
```

Pentru primul format, numele de gazdă prestabilit este `localhost`. (Există şi alte formate permise pentru sursele de date, despre care vom discuta ulterior în secţiunea „Specificarea parametrilor de conexiune") Nu contează mărimea literelor folosite în "`DBI`", dar "`mysql`" trebuie scris cu litere mici.

- **Numele dumneavoastră de utilizator şi parola.**
- **Un argument opţional, care indică atribute de conexiune suplimentare.** Acesta controlează modalitatea DBI de tratare a erorilor, iar construcţia cu aspect ciudat pe care am specificat-o activează atributul `RaiseError`. Aceasta determină modulul DBI să caute erorile relative la bazele de date, respectiv să afişeze un mesaj şi să-şi încheie execuţia ori de câte ori întâlneşte o eroare. (Iată de ce nu vedeţi nici un program de detecţie a apariţiei erorilor nicăieri în interiorul scriptului `dump_members`; DBI se ocupă de această operaţie.) Secţiunea „Tratarea erorilor" discută despre metode alternative de reacţie la apariţia unor erori.

În cazul în care apelul la funcţia `connect()` reuşeşte, returnează o variabilă de manipulare pentru baze de date, pe care o atribuim lui `$dbh`. (Dacă apelul la `connect()` eşuează, în mod normal returnează `undef`. Totuşi, deoarece am activat `RaiseError` în scriptul nostru, `connect()` nu va returna; în schimb, DBI va afişa un mesaj de eroare şi îşi va încheia execuţia în cazul apariţiei unei erori.)

După conectarea la baza de date, `dump_members` emite o interogare `SELECT` pentru a regăsi lista membrilor, apoi execută un ciclu pentru a prelucra fiecare din rândurile returnate. Aceste rânduri constituie setul de rezultate.

Pentru a efectua o interogare `SELECT`, mai întâi o pregătiţi şi apoi o executaţi:

```
# emiterea interogarii
$sth = $dbh->prepare ("SELECT nume, prenume, sufix, email,"
    . "strada, oras, stat, cod_postal, telefon FROM membru"
    " ORDER BY nume");
$sth->execute ();
```

Funcţia `prepare()` este apelată folosind variabila de manipulare pentru baze de date; aceasta transferă driverului instrucţiunea SQL pentru pre-procesare anterior execuţiei. Unele drivere utilizează efectiv instrucţiunea în acest punct. Altele îşi reamintesc de instrucţiune doar atunci când invocaţi `execute()` pentru a determina efectuarea instrucţiunii. Valoarea returnată de `prepare()` este o variabilă de manipulare pentru instrucţiuni, adică `$sth`, respectiv `undef` în cazul apariţiei unor erori. Variabila de manipulare pentru instrucţiuni este folosită pentru toate prelucrările ulterioare legate de instrucţiune.

Reţineţi că interogarea este specificată fără caracterul de terminare punct şi virgulă. Categoric, aveţi obiceiul (dezvoltat după un timp îndelungat de lucru cu programul `mysql`) de a termina instrucţiunile SQL cu un caracter `;`. Totuşi, cel mai bine este să vă dezbăraţi de acest obicei atunci când folosiţi DBI, deoarece deseori caracterele punct şi virgulă determină eşecul interogărilor datorită unor erori de sintaxă. Aceeaşi regulă se aplică şi pentru caracterele `\g`: nu le adăugaţi la sfârşitul interogărilor!

Când invocaţi o metodă fără a-i transfera nici un argument, puteţi omite parantezele. Următoarele două apeluri sunt echivalente:

```
$sth->execute ();
$sth->execute;
```

Eu prefer să folosesc parantezele, deoarece astfel apelul seamănă mai puţin cu o referinţă la o variabilă. Poate că preferinţele dumneavoastră sunt altele.

După ce apelaţi funcţia `execute()`, rândurile din lista cu membri pot fi prelucrate. În scriptul `dump_members`, ciclul de preluare a rândurilor pur şi simplu afişează conţinutul fiecărui rând:

```
# citeste rezultatele interogarii, apoi face curatenie
while (@ary = $sth->fetchrow_array())
{
    print join ("\t", @ary), "\n"
}
$sth->finish ();
```

Funcţia `fetchrow_array()` returnează un tablou care conţine valorile coloanelor din rândul curent, respectiv un tablou vid atunci când nu mai există rânduri. Astfel, ciclul preia rândurile succesive returnate de instrucţiunea `SELECT` şi îl afişează pe fiecare, inserând tabulatori între valorile coloanelor. Valorile `NULL` din baza de date sunt returnate scriptului Perl ca valori `undef`, dar acestea sunt afişate ca şiruri vide, nu sub forma cuvântului `NULL`.

Observaţi includerea tabulatorilor şi a caracterelor linie nouă (reprezentate sub forma `'\t'` respectiv `'\n'`) între ghilimele duble. În Perl, secvenţele escape sunt interpretate numai când apar între ghilimele duble, nu între ghilimele simple. Dacă au fost folosite ghilimelele simple, rezultatul va fi plin de instanţe literale ale şirurilor `"\t"` şi `"\n"`.

După terminarea ciclului de preluare a rândurilor, apelul la funcţia `finish()` indică programului DBI că variabila de manipulare pentru instrucţiuni nu mai este necesară şi că pot fi eliberate toate resursele temporare asociate acesteia. Apelarea funcţiei `finish()` nu este necesară decât dacă aţi preluat numai o parte din setul de rezultate (prin proiectare sau datorită apariţiei anumitor probleme). Totuşi, `finish()` poate fi întotdeauna utilizat în siguranţă după un ciclu de preluare şi mi se pare mai uşor să apelaţi funcţia şi să o utilizaţi decât să scrieţi logica necesară pentru a distinge între situaţiile în care `finish()` este necesară de cele în care funcţia respectivă este superfluă.

După ce am afişat lista membrilor, am terminat, deci ne putem deconecta de la server şi încheia execuţia programului:

```
$dbh->disconnect ();
exit (0);
```

Scriptul `dump_members` ilustrează un număr de concepte care sunt comune majorităţii programelor DBI, iar dumneavoastră puteţi începe probabil să vă scrieţi propriile programe DBI fără a avea alte cunoştinţe suplimentare. De exemplu, pentru a scrie conţinutul unui alt tabel, tot ce aveţi de făcut este să modificaţi textul instrucţiunii `SELECT` care este transferată metodei `prepare()` De fapt, dacă doriţi să vedeţi unele aplicaţii ale acestei tehnici, puteţi trece imediat la acea parte a secţiunii „Utilizarea DBI" care discută

despre modul de generare a listei membrilor pentru programul banchetului anual al Ligii istorice, precum și despre modul de generare a catalogului tipărit al Ligii. Totuși, DBI furnizează multe alte caracteristici utile. Secțiunile următoare tratează despre unele din acestea într-un mod mai detaliat, astfel încât să puteți utiliza scripturile Perl nu numai pentru rularea unor simple instrucțiuni `SELECT`.

Tratarea erorilor

Scriptul `dump_members` a activat atributul de tratare a erorilor `RaiseError` atunci când a invocat metoda `connect()`, astfel încât erorile să cauzeze terminarea imediată a execuției scriptului cu un mesaj de eroare. Există și alte metode de tratare a erorilor. De exemplu, puteți verifica personal apariția erorilor în loc de a cere programului DBI să execute această operație.

Pentru a vedea cum se poate controla modul de tratare a erorilor în DBI, să examinăm mai îndeaproape argumentul final al funcției `connect()`. Cele două atribute relevante sunt `RaiseError` și `PrintError`:

- Dacă `RaiseError` este activat (adică primește o valoare diferită de zero), DBI apelează funcția `die()` pentru a tipări un mesaj și pentru a-și termina execuția dacă se produce vreo eroare într-o metodă DBI.
- Dacă `PrintError` este activat, DBI apelează funcția `warn()` pentru a afișa un mesaj atunci când se produce o eroare DBI, dar scriptul își continuă execuția.

În mod prestabilit, `RaiseError` este dezactivat și `PrintError` este activat. În acest caz, dacă apelul la funcția `connect()` eșuează, DBI afișează un mesaj, dar își continuă execuția. Astfel, folosind metoda prestabilită de tratare a erorilor, pe care o obțineți dacă omiteți al patrulea argument al funcției `connect()`, puteți căuta erori în acest mod:

```
$dbh = DBI->connect ($dsn, $user_name, $password)
    or exit (1);
```

Dacă se produce vreo eroare, funcția `connect()` returnează `undef` pentru a indica eșecul, ceea ce declanșează apelul la `exit()`. Nu trebuie să afișați un mesaj de eroare, deoarece DBI va fi afișat deja unul.

Dacă specificați în mod explicit valorile prestabilite ale atributelor operației de verificare a erorilor, apelul la funcția `connect()` se prezintă astfel:

```
$dbh = DBI->connect ($dsn, $user_name, $password,
                     { RaiseError => 0, PrintError => 1})
    or exit (1);
```

Este ceva mai mult de scris, dar un cititor ocazional va sesiza mai rapid care este metoda de tratare a erorilor utilizată.

Dacă doriți să căutați personal erorile și să vă afișați propriile mesaje, dezactivați atât `RaiseError`, cât și `PrintError`:

```
$dbh = DBI->connect ($dsn, $user_name, $password,
                     { RaiseError => 0, PrintError => 0})
    or die "Nu se poate conecta la server: $DBI::err ($DBI::errstr)\n";
```


Variabilele `$DBI::err` și `$DBI::errstr`, folosite în apelul la funcția `die()` prezentat anterior, sunt utile pentru construcția mesajelor de eroare. Aceste variabile conțin codul de eroare și șirul de eroare, asemănător cu funcțiile `mysql_errno()`, respectiv `mysql_error()` din interfața API pentru C.

Dacă nu doriți decât ca DBI să trateze erorile în locul dumneavoastră, astfel încât să nu fie necesar să le verificați personal, activați `RaiseError`:

```
$dbh = DBI->connect ($dsn, $user_name, $password, { RaiseError => 1 });
```

Aceasta reprezintă de departe metoda cea mai simplă și este cea folosită de scriptul `dump_members`. Activarea atributului `RaiseError` poate fi inadecvată dacă doriți să executați un program personal de curățenie atunci când scriptul își încheie execuția, deși în acest caz puteți efectua operațiile dorite redefinind variabila de manipulare `$SIG{__DIE__}`.

Un alt motiv pentru care doriți să evitați activarea atributului `RaiseError` este acela că DBI afișează informații de natură tehnică în mesajele sale, după cum urmează:

```
disconnect(DBI::db=HASH(0x197aae4)) invalidates 1 active statement. Either
destroy statement handles or call finish on them before disconnecting.
```

(Instrucțiunea ... invalidează 1 instrucțiune activă. Distrugeți variabilele de manipulare pentru instrucțiuni sau apelați funcția `finish()` pentru acestea înainte de deconectare.)

Acestea constituie informații utile pentru un programator, dar nu este genul de lucruri pe care utilizatorul cotidian dorește să le vadă. În acest caz, este mai bine să verificați personal apariția erorilor, astfel încât să puteți afișa mesaje mai semnificative pentru persoanele care vor utiliza scriptul. Alternativ, puteți lua în considerare redefinirea variabilei de manipulare `$SIG{__DIE__}`. Acest lucru poate fi util, deoarece vă permite să activați `RaiseError` pentru simplificarea procedurii de tratare a erorilor, dar și să înlocuiți mesajele de eroare prestabilite pe care le prezintă DBI cu propriile dumneavoastră mesaje. Pentru a vă furniza propria variabilă de manipulare `__DIE__`, înainte de a executa vreun apel DBI, procedați astfel:

```
$SIG{__DIE__} = sub {die "Scuze, s-a produs o eroare\n"; };
```

De asemenea, puteți declara o subrutină în maniera uzuală și puteți stabili valoarea variabilei de manipulare folosind o referință la subrutină:

```
sub die_handler
{
    die "Scuze, s-a produs o eroare\n";
}

$SIG{__DIE__} = \&die_handler;
```

Ca alternativă la transferul în formă literală al atributelor operației de tratare a erorilor în cadrul apelului la funcția `connect()`, acestea pot fi definite folosind o funcție hash și transferând o referință la hash. Unii sunt de părere că „extinderea" valorilor atributelor în acest mod facilitează citirea și editarea scripturilor, dar cele două metode sunt identice din punct de vedere funcțional. Iată un exemplu care prezintă modul de utilizare al valorii hash al unui atribut:

```
%attr=
(
   PrintError => 0,
   RaiseError => 0
);
$dbh = DBI->connect ($dsn, $user_name, $password, \%attr)
      or die "Nu se poate conecta la server: $DBI::err ($DBI::errstr)\n";
```

Scriptul următor, intitulat `dump_members2`, ilustrează modul de scriere a unui script atunci când doriţi să verificaţi personal apariţia erorilor şi să vă scrieţi propriile mesaje. `dump_members2` prelucrează aceeaşi interogare ca şi `dump_members`, dar dezactivează în mod explicit `PrintError` şi `RaiseError`, după care testează rezultatul fiecărui apel DBI. Când se produce o eroare, scriptul invocă subrutina `bail_out()` care afişează un mesaj, respectiv conţinutul variabilelor `$DBI::err` şi `$DBI::errstr` înainte de a-şi încheia execuţia:

```
#! /usr/bin/perl

# dump_members2 - afiseaza lista membrilor Ligii istorice

use DBI;
use strict;

my ($dsn) = "DBI:mysql:samp_db:localhost"; # nume sursa date
my ($user_name) = "paul";     # numele utilizatorului
my ($password) = "secret"     # parola
my ($dbh, $sth);              # variabile pentru baza de date si
                                instructiune
my (@ary);                    # tablou pentru randurile returnate de
                                interogare
my (%attr) =                  # atributele de tratare a erorilor
(
   PrintError => 0,
   RaiseError => 0
);

# conectarea la baza de date
$dbh = DBI->connect ($dsn, $user_name, $password, \%attr)
   or bail_out ("Nu se poate conecta la baza de date");

# emiterea interogarii
$sth = $dbh->prepare ("SELECT nume, prenume, sufix, email,"
   . "strada, oras, stat, cod_postal, telefon FROM membru"
   "ORDER BY nume")
   or bail_out ("Nu poate prepara interogarea");
$sth->execute ()
   or bail_out ("Nu poate executa interogarea");
```



```
# citeste rezultatele interogarii
while (@ary = $sth->fetchrow_array())
{
   print join ("\t", @ary), "\n";
}
$DBI::err == 0
   or bail_out ("Eroare in timpul regasirii");

# curatenie
$sth->finish ()
   or bail_out ("Nu poate finaliza interogarea");

$dbh->disconnect ()
   or bail_out ("Nu se poate deconecta de la baza de date");
exit (0);

# subrutina bail_out() - afiseaza cod de eroare si sirul de eroare,
                         apoi isi incheie executia

sub bail_out
{
my ($message) = shift;

   die "$message\nError $DBI::err ($DBI::errstr)\n");
}
```

Funcţia `bail_out()`este similară cu funcţia `print_error()` pe care am folosit-o pentru scrierea programelor C din capitolul 6, cu deosebirea că `bail_out()` îşi încheie execuţia, nu returnează controlul apelantului. `bail_out()` vă scuteşte de efortul de a scrie numele variabilelor `$DBI::err` şi `$DBI::errstr` de fiecare dată când doriţi să scrieţi un mesaj de eroare. De asemenea, prin încapsularea afişării mesajelor într-o subrutină, puteţi schimba în mod uniform formatul mesajelor dumneavoastră de eroare în întreg scriptul, efectuând o modificare în subrutină.

Scriptul `dump_members` include după ciclul de preluare a rândurilor un test pe care scriptul `dump_members` nu-l conţine. Deoarece `dump_members2` nu-şi încheie automat execuţia dacă se produce vreo eroare în tabloul `fetchrow_array()`, este mai prudent să se determine dacă ciclul s-a terminat datorită citirii complete a setului de înregistrări (terminare normală) sau datorită producerii unei erori. Ciclul se încheie în ambele situaţii, desigur, dar, în cazul apariţiei unei erori, datele de ieşire ale scriptului vor fi trunchiate. În absenţa unei verificări a apariţiei erorilor, persoana care rulează scriptul nu va avea nici cel mai mic indiciu că s-ar fi întâmplat ceva! Dacă verificaţi personal apariţia erorilor, nu uitaţi să testaţi rezultatul ciclurilor de preluare.

Tratarea interogărilor care nu returnează nici un set de rezultate

Instrucțiuni precum `DELETE`, `INSERT`, `REPLACE` și `UPDATE`, care nu returnează rânduri, sunt relativ simplu de prelucrat în comparație cu instrucțiuni precum `SELECT`, `DESCRIBE`, `EXPLAIN` și `SHOW`, care returnează rânduri. Pentru a prelucra o instrucțiune care nu este de tip `SELECT`, transferați-o funcției `do()` folosind variabila de manipulare pentru baze de date. Metoda `do()` prepară și execută interogarea într-o singură etapă. De exemplu, pentru a introduce rubrica aferentă unui membru nou, pe nume Cornel Vasile, cu data expirării 3 iunie 2002, puteți proceda astfel:

```
$rows = $dbh->do ("INSERT membru (nume,prenume,expirare)"
               . " VALUES('Vasile','Cornel','2002-6-3')");
```

Metoda `do()` returnează un număr al rândurilor afectate, respectiv `undef` dacă se produce vreo eroare. O eroare se poate produce din diverse motive. (De exemplu, interogarea este deformată sau dumneavoastră nu aveți permisiunea de a obține acces la tabel.) Pentru o valoare returnată diferită de `undef`, fiți atent la situația în care nu este afectat nici un rând. Când survine o atare situație, funcția `do()` nu returnează numărul `0`; în schimb, returnează șirul `"0E0"` (notația științifică folosită în Perl pentru zero). `"0E0"` are valoarea `0` într-un context numeric, dar este considerat ca „adevărat" în teste condiționale, astfel încât să poată fi deosebit cu ușurință de `undef`. Dacă funcția `do()` ar fi returnat `0`, atunci ar fi fost mai dificil să se facă diferența dintre apariția unei erori (`undef`) și cazul în care nu a fost afectat nici un rând. Puteți verifica apariția unei erori folosind oricare din următoarele teste:

```
if (!defined ($rows)) { # eroare }
if (!$rows) { # eroare }
```

În contexte numerice, `"0E0"` are valoarea `0`. Programul următor va afișa corect numărul rândurilor pentru orice valoare a variabilei `$rows` diferită de `undef`:

```
if (!$rows)
{
   print "eroare\n";
}
else
{
   $rows += 0; # conversie fortata la numar daca "0E0"
   print "$rows randuri afectate\n";
}
```

De asemenea, puteți afișa valoarea variabilei `$rows` folosind un format `%d` cu funcția `printf()` pentru a forța o conversie implicită la un număr:

```
if (!$rows)
{
   print "eroare\n";
}
else
{
   print "%d $rows randuri afectate\n", $rows;
}
```


Metoda `do()` este echivalentă cu `prepare()`, urmată de `execute()`. Instrucțiunea `INSERT` anterioară poate fi emisă și astfel, nu numai prin apelarea funcției `do()`:

```
$sth = $dbh->prepare("INSERT membru (nume,prenume,expirare)"
                   . " VALUES('Vasile','Cornel','2002-6-3')");
$rows = $sth->execute ();
```

Tratarea interogărilor care returnează un set de rezultate

Această secțiune furnizează mai multe informații despre numeroase opțiuni pe care le aveți la dispoziție în vederea executării ciclului de preluare a rândurilor pentru interogările `SELECT` (sau pentru alte interogări de tip `SELECT` care returnează rânduri, precum `DESCRIBE`, `EXPLAIN` și `SHOW`). De asemenea, se discută despre obținerea numărului de rânduri ale unui rezultat, modul de tratare a seturilor de rezultate pentru care nu este necesar nici un ciclu, respectiv despre regăsirea „dintr-o mișcare" a unui întreg set de rezultate.

Scrierea ciclurilor de preluare a rândurilor

Scriptul `dump_members` regăsea date folosind o secvență standard de metode DBI: `prepare()` pentru a permite driverului să pre-proceseze interogarea, `execute()` pentru a începe execuția interogării, `fetchrow_array()` pentru a prelua fiecare rând al setului de rezultate, respectiv `finish()` pentru a elibera resursele alocate interogării.

`prepare()`, `execute()` și `finish()` sunt componente oarecum standard ale prelucrării oricărei interogări care returnează rânduri. Totuși, pentru preluarea rândurilor, `fetchrow_array()` este numai o opțiune din alte numeroase metode (vezi tabelul 7.3).

Tabelul 7.3 Metode DBI pentru preluarea rândurilor

Numele metodei	Valoare returnată
`fetchrow_array()`	Tablou cu valorile din rând
`fetchrow_arrayref()`	Referință la un tablou cu valorile din rând
`fetch()`	Similar cu `fetchrow_arrayref()`
`fetchrow_hashref()`	Referință la valorile hash ale valorilor din rând, care conțin și numele coloanei drept cheie

Exemplele următoare prezintă modul de utilizare a fiecărei metode de preluare a rândurilor. Exemplele parcurg ciclic rândurile unui set de rezultate și, pentru fiecare rând, afișează valorile din coloane separate prin virgule. Există metode mai eficiente de a scrie codul de afișare în anumite situații, dar exemplele sunt astfel scrise pentru a ilustra sintaxa necesară accesului la valorile individuale din coloane.

`fetchrow_array()` este folosită după cum urmează:

```
while (@ary = $sth->fetchrow_array ())
{
   $delim = "";
```

Continuare

Continuare

```
    for ($i = 0; $i < @ary; $i++)
    {
        print $delim . $ary[$i];
        $delim = ",";
    }
    print "\n";
}
```

Fiecare apel la funcția `fetchrow_array()` returnează un tablou de valori din rând, respectiv un tablou vid dacă nu mai există rânduri.

Ca alternativă la atribuirea valorii returnate unei variabile de tip tablou, puteți prelua valorile din coloane într-un set de variabile scalare. Puteți face aceasta dacă doriți să lucrați cu nume de variabile care sunt mai semnificative decât `$ary[0]`, `$ary[1]` etc. Să presupunem că doriți să regăsiți valorile numelor și ale adreselor de e-mail în variabile. Folosind `fetchrow_array()`, puteți selecta și prelua rânduri astfel:

```
$sth = $dbh->prepare ("SELECT nume, prenume, sufix, email,"
    . "strada, oras, stat, cod_postal, telefon FROM membru"
    . "ORDER BY nume");
$sth->execute ();
while (($nume, $prenume, $sufix, $email) = $sth->fetchrow_array())
{
    # se executa operatii cu variabilele
}
```

Când folosiți o listă de variabile în acest mod, trebuie să vă asigurați că interogarea dumneavoastră selectează coloanele în ordinea corectă, desigur. DBI nu cunoaște ordinea în care sunt denumite coloanele în instrucțiunea dumneavoastră `SELECT`, deci dumneavoastră trebuie să atribuiți variabilele corect. De asemenea, puteți determina ca variabilele individuale să primească în mod automat valorile coloanelor atunci când selectați un rând, folosind o tehnică cunoscută sub numele de asociere a parametrilor. Despre aceasta vom discuta mai detaliat în secțiunea „Câmpuri de înlocuire și asocierea parametrilor".

`fetchrow_arrayref()` este similară cu `fetchrow_array()` dar, în loc de a returna un tablou care conține valorile coloanelor pentru rândul curent, pur și simplu returnează o referință la tablou, respectiv `undef` atunci când nu mai există rânduri. Se folosește astfel:

```
while ($ary_ref = $sth->fetchrow_arrayref())
{
    $delim = "";
    for ($i = 0; $i < @{$ary_ref}; $i++)
    {
        print $delim . $ary_ref->[$i];
        $delim = ",";
    }
    print "\n";
}
```


Elementele tabloului sunt accesibile prin intermediul referinței la tablou, `$ary_ref`. Aceasta este oarecum asemănător cu determinarea entității indicate de un pointer, deci folosiți `$ary_ref->[$i]` în loc de `$ary[$i]`. Pentru a trata referința ca tablou complet, folosiți construcția `@{$ary_ref}`.

`fetchrow_arrayref()` nu este adecvată pentru preluarea variabilelor într-o listă. De exemplu, ciclul următor nu funcționează:

```
while (($nume, $prenume, $sufix $email) = @{$sth->fetchrow_arrayref()})
{
    # se utilizeaza variabilele intr-un anumit mod
}
```

Atâta timp cât `fetchrow_arrayref()` preia efectiv un rând, ciclul funcționează corect. Când însă rândurile se termină, `fetchrow_arrayref()` returnează `undef`, iar `@{undef}` nu este o variabilă corectă. (Este similar cu încercarea de determinare a entității indicate de un pointer `NULL` într-un program C.)

Cea de-a treia metodă de preluare a rândurilor, `fetchrow_hashref()`, se folosește astfel:

```
while ($hashref = $sth->fetchrow_hashref())
{
    $delim = "";
    foreach $key (keys (%{$hashref}))
    {
        print $delim . $hashref->{$key};
        $delim = ",";
    }
    print "\n";
}
```

Fiecare apel la funcția `fetchrow_hashref()` returnează o referință la un hash al valorilor unui rând, ordonate după numele coloanelor, respectiv `undef` dacă nu mai sunt rânduri. În acest caz, valorile coloanelor nu sunt dispuse într-o ordine anumită; membrii funcțiilor hash din Perl nu sunt ordonați. Cu toate acestea, elementele hash sunt ordonate în funcție de numele coloanei, deci `$hashref` vă oferă o singură variabilă prin intermediul căreia puteți obține acces la orice valoare de coloană, în funcție de nume. Aceasta vă permite să extrageți valori (sau orice subset de valori) și nu trebuie să cunoașteți ordinea în care interogarea `SELECT` a regăsit coloanele. De exemplu, dacă doriți să obțineți acces la câmpurile cu numele și adresa de e-mail, procedați astfel:

```
while ($hashref = $sth->fetchrow_hashref())
{
    $delim = "";
    foreach $key ("nume", "prenume", "sufix", "email")
    {
        print $delim . $hashref->{$key};
        $delim = ",";
    }
    print "\n";
}
```

Funcţia `fetchrow_hashref()` este utilă mai ales când doriţi să transferaţi unei funcţii un rând de valori, fără a impune funcţiei să cunoască ordinea în care au fost denumite coloanele în instrucţiunea `SELECT`. În acest caz, veţi apela funcţia `fetchrow_hashref()` pentru regăsirea rândurilor şi veţi scrie o funcţie care obţine acces la valorile din hash-ul rândului, folosind numele coloanelor.

Reţineţi următoarele pericole atunci când folosiţi `fetchrow_hashref()`:

- Dacă aveţi nevoie de maximum de performanţă, `fetchrow_hashref()` nu este cea mai bună opţiune, deoarece nu este la fel de eficientă ca `fetchrow_array()` sau `fetchrow_arrayref()`.
- Numele coloanelor folosite ca valori cheie în hash respectă mărimea literelor folosite la scrierea coloanelor în instrucţiunea `SELECT`. În MySQL, numele coloanelor nu sunt sensibile la diferenţa între majuscule şi minuscule, deci interogarea va avea acelaşi efect, indiferent de mărimea literelor folosite pentru scrierea numelor coloanelor. Dar numele elementelor hash cheie din Perl sunt sensibile la diferenţa între majuscule şi minuscule, fapt ce vă poate provoca probleme. Pentru a evita posibile probleme legate de neconcordanţe între mărimile literelor, puteţi cere funcţiei `fetchrow_hashref()` să forţeze scrierea numelor coloanelor folosind o anumită mărime de literă, transferându-i acesteia un atribut `NAME_lc` sau `NAME_uc`:

```
$hash_ref = $sth->fetchrow_hashref ('NAME_lc');
    # foloseste nume scrise cu minuscule
$hash_ref = $sth->fetchrow_hashref ('NAME_uc');
    # foloseste nume scrise cu majuscule
```

- Hash-ul conţine câte un element pentru fiecare nume distinct de coloană. Dacă efectuaţi o unire care returnează coloane din mai multe tabele cu nume care se suprapun, nu veţi putea avea acces la toate valorile din coloane. De exemplu, dacă emiteţi următoarea interogare, `fetchrow_hashref()` va returna un hash având un singur element:

```
SELECT a.nume, b.nume FROM a, b, WHERE a.nume = b.nume
```

Determinarea numărului de rânduri returnate de o interogare

Cum puteţi determina numărul de înregistrări returnate de o interogare `SELECT` sau de tip `SELECT`? O modalitate este aceea de a număra rândurile pe măsură ce le preluaţi, desigur. De fapt, aceasta este singura metodă portabilă de a cunoaşte numărul de rânduri pe care le returnează o interogare `SELECT`. Utilizând driverul MySQL, puteţi apela metoda `rows()` folosind variabila de manipulare a instrucţiunii după invocarea funcţiei `execute()`, dar acest procedeu nu este portabil la alte motoare de baze de date. Nici măcar în MySQL `rows()` nu returnează rezultatul corect decât atunci când aţi preluat toate rândurile, dacă aţi activat atributul `mysql_use_result`. (Pentru mai multe informaţii, vezi Anexa G.) Deci, puteţi la fel de bine număra rândurile pe măsură ce le preluaţi.

Preluarea rezultatelor compuse dintr-un singur rând

Nu este necesar să rulaţi un ciclu pentru obţinerea rezultatelor, dacă setul de rezultate constă dintr-un singur rând. Să presupunem că doriţi să scrieţi un script `count_members`, care vă indică numărul curent al membrilor Ligii istorice. Codul pentru efectuarea interogării are următorul aspect:


```
# emite interogarea
$sth = $dbh->prepare ("SELECT COUNT(*) FROM membru");
$sth->execute ();

# citeste rezultatele interogarii, apoi face curatenie
$count = $sth->fetchrow_array ();
$sth->finish ();
$count = "nedeterminat" if !defined ($count);
print "$count\n";
```

Instrucţiunea `SELECT` va returna un singur rând, deci nu este necesar un ciclu; vom apela funcţia `fetchrow_array()` numai o dată. În plus, deoarece selectăm o singură coloană, nu este necesară nici măcar atribuirea valorii returnate unui tablou. Când funcţia `fetchrow_array()` este apelată într-un context scalar (unde se aşteaptă o singură valoare, nu o listă întreagă), va returna valoarea din prima coloană a rândului, respectiv `undef` dacă nu mai există rânduri disponibile.

Un alt tip de interogări pentru care este de aşteptat cel mult o singură înregistrare este cel care conţine `LIMIT 1`, pentru limitarea numărului de rânduri returnate. O utilizare comună a acestui atribut este pentru returnarea rândului care conţine valoarea minimă, respectiv maximă a unei anumite coloane. De exemplu, interogarea următoare afişează numele şi data naşterii preşedintelui american care s-a născut cel mai recent:

```
$query = "SELECT nume, prenume, data_nastere"
    . " FROM presedinte ORDER BY data_nastere DESC LIMIT 1";
$sth = $dbh->prepare ($query);
$sth->execute ();

# citeste rezultatele interogarii, apoi face curatenie
($nume, $prenume, $data_nastere) = $sth->fetchrow_array ();
$sth->finish ();
if (!defined ($nume))
{
    print "Interogarea nu a returnat nici un rezultat\n";
}
else
{
    print "Presedintele nascut cel mai recent:
      $nume $prenume ($data_nastere)\n";
}
```

Alte tipuri de interogări pentru care nu este necesar nici un ciclu de preluare sunt cele care folosesc `MAX()` sau `MIN()` pentru a selecta o singură valoare. Dar, în toate aceste situaţii, o modalitate chiar mai simplă de a obţine un rezultat cu un singur rând este de a folosi metoda cu variabila de manipulare pentru baze de date `selectrow_array()`, care combină funcţiile `prepare()`, `execute()` şi rutina de preluare a rândurilor într-un singur apel. Funcţia `selectrow_array()` returnează un tablou (nu o referinţă), respectiv un

tablou vid în cazul apariției unor erori. Exemplul anterior poate fi scris astfel, folosind funcția `selectrow_array()`:

```
$query = "SELECT nume, prenume, data_nastere"
   . " FROM presedinte ORDER BY data_nastere DESC LIMIT 1";
($nume, $prenume, $data_nastere) = $sth->selectrow_array ($query);
if (!defined ($nume))
{
   print "Interogarea nu a returnat nici un rezultat\n";
}
else
{
   print "Presedintele nascut cel mai recent:
      $nume $prenume ($data_nastere)\n";
}
```

Lucrul cu seturi de rezultate complete

Când folosiți un ciclu de preluare, DBI nu furnizează vreo modalitate de căutare aleatoare în setul de prelucrare și nici o metodă de prelucrare a rândurilor în orice altă ordine decât cea în care le-a returnat ciclul. De asemenea, după ce preluați un rând, rândul anterior se pierde, dacă nu luați măsuri pentru a-l păstra în memorie. Această comportare nu este întotdeauna adecvată:

- Puteți dori să prelucrați rânduri într-o ordine non-secvențială. Să considerăm o situație când doriți să alcătuiți un chestionar care se bazează pe datele despre președinții americani menționate în tabelul `presedinte` al Ligii istorice. Dacă doriți să puneți întrebări într-o altă ordine la fiecare prezentare a chestionarului, puteți selecta toate rândurile din tabelul `presedinte`. Apoi, puteți selecta rânduri într-o ordine aleatoare, pentru a varia ordinea președinților despre care puneți întrebările. Pentru a selecta un rând în mod aleator, trebuie să aveți acces simultan la toate rândurile.
- Puteți dori să folosiți numai un subset al rândurilor returnate, subset selectat aleator. De exemplu, pentru a prezenta o întrebare cu opțiuni multiple cu privire la locul nașterii unui președinte, puteți selecta aleator un rând pentru a selecta președintele (și răspunsul corect), iar apoi alegeți alte rânduri din care să extrageți opțiunile eronate.
- Puteți dori să obțineți întregul set de rezultate, chiar dacă îl prelucrați în ordine secvențială. Acest lucru poate fi necesar dacă trebuie să executați mai multe parcurgeri ale rândurilor. De exemplu, într-un calcul statistic, puteți parcurge o dată setul de rezultate pentru a evalua unele proprietăți numerice generale ale datelor dumneavoastră, iar apoi parcurgeți rândurile din nou, executând o analiză cu un caracter mai specific.

Puteți obține acces la setul de rezultate în totalitatea sa în două moduri diferite. Puteți efectua ciclul de preluare obișnuit și salvați fiecare rând pe măsură ce îl preluați, fie puteți folosi o metodă care returnează un întreg set de rezultate dintr-o dată. Indiferent cum procedați, veți obține o matrice care conține un rând pentru fiecare rând al setului de rezultate, respectiv numărul de coloane pe care le-ați selectat. Puteți prelucra elementele matricei în orice ordine doriți, de oricâte ori doriți. Expunerea următoare descrie ambele metode.

O modalitate de a folosi un ciclu de preluare pentru capturarea setului de rezultate este de a folosi funcția `fetchrow_array()` și de a salva un tablou cu referințe la rânduri. Programul următor procedează într-un mod similar cu ciclul de preluare-afișare din scriptul `dump_members`, cu deosebirea că salvează toate rândurile, apoi afișează matricea pentru a ilustra modul de determinare a numărului de rânduri și coloane din matrice, precum și modul de acces la membrii individuali ai matricei.

```
my (@matrix) = (); # tablou de referinte la tablou

while (my @ary = $sth->fetchrow_array ()) # preia fiecare rand
{
   push (@matrix, [ @ary ]); # salveaza referinta la randul preluat
}
$sth->finish ();

# determina dimensiunile matricei
my (@rows) = scalar (@matrix);
my (@cols) = ($rows == 0 ? 0 : scalar (@{$matrix[0]}));

for (my $i = 0; $i < $rows ; $i++)     # afiseaza fiecare rand
{
my ($delim) = "";
   for (my $j = 0; $j < $cols ; $j++)
   {
      print $delim . $matrix[$i][$j];
      $delim = ",";
   }
   print "\n";
}
```

Când stabiliți dimensiunile matricei, numărul rândurilor trebuie să fie determinat primul, deoarece calculul numărului de coloane depinde de faptul dacă matricea este vidă sau nu. Dacă `$rows` este 0, matricea este vidă și `$cols` devine de asemenea 0. În caz contrar, numărul coloanelor poate fi calculat ca fiind numărul elementelor din tabloul aferent unui rând, folosind sintaxa `@{$matrix[$i]}` pentru a avea acces la rândul `$i` în totalitatea sa.

În exemplul anterior, am preluat fiecare rând, apoi am salvat o referință la acesta. Ați putea presupune că este mai eficient să apelați funcția `fetchrow_arrayref()` în loc să regăsiți direct referințele la rânduri:

```
my (@matrix) = (); # tablou de referinte la tablou

while (my @ary_ref =$sth->fetchrow_arrayref ())
{
   # acest procedeu nu functioneaza
   push (@matrix, @ary_ref); # salveaza referinta la randul preluat
}
$sth->finish ();
```

Metoda de mai sus nu funcționează, deoarece `fetchrow_arrayref()` refolosește tabloul spre care indică referința. Matricea rezultantă este un tablou de referințe, unde fiecare referință indică spre același rând – rândul regăsit ultimul. Ca atare, dacă doriți să preluați rând cu rând, folosiți `fetchrow_array()` în loc de `fetchrow_arrayref()`.

Ca alternativă la utilizarea unui ciclu de preluare, puteți folosi una dintre metodele DBI care returnează întregul set de rezultate. De exemplu, `fetchall_arrayref()` returnează o referință la un tablou de referințe, unde fiecare din aceste referințe indică spre conținutul unui rând al setului de rezultate. Este puțin, dar, ca efect, valoarea returnată este o referință la o matrice. Pentru a folosi `fetchall_arrayref()`, apelați `prepare()` și `execute()`, apoi regăsiți rezultatul astfel:

```
my ($matrix_ref); # referinta la un tablou de referinte

$matrix_ref = $sth->fetchall_arrayref (); # preia toate randurile

# determina dimensiunile matricei
my ($rows) = (!defined ($matrix_ref) ? 0 : scalar (@{$matrix_ref}));
my ($cols) = ($rows == 0 ? 0 : scalar (@{$matrix_ref->[0]}));

for (my $i = 0; $i < $rows; $i++)      # afiseaza fiecare rand
{
my ($delim) = "";
   for (my $j = 0; $j < $cols; $j++)
   {
      print $delim . $matrix_ref->[$i][$j];
      $delim = ",";
   }
   print "\n";
}
```

Funcția `fetchall_arrayref()` returnează o referință la un tablou vid, dacă setul de rezultate este vid. Rezultatul este `undef` dacă survine o eroare deci, dacă atributul `RaiseError` nu este activat, nu uitați să verificați valoarea returnată înainte de a începe să o folosiți.

Numărul de rânduri și de coloane este determinat de faptul dacă matricea este sau nu vidă. Dacă doriți să obțineți acces la un întreg rând `$i` al matricei ca la un tablou, folosiți sintaxa `@{$matrix_ref->[i]}`.

Este categoric mai simplu să se folosească `fetchall_arrayref()` pentru a regăsi un set de rezultate decât să se scrie un ciclu de preluare a rândurilor, deși sintaxa pentru accesul la elementele tabloului devine puțin mai complicată. O metodă similară cu `fetchall_arrayref()`, dar care execută mai multe operații, este `selectall_arrayref()`. Această metodă execută automat întreaga secvență formată din `prepare()`, `execute()`, ciclul de preluare și `finish()`. Pentru a folosi `selectall_arrayref()`, transferați interogarea direct acestei funcții, folosind variabila de manipulare pentru baze de date:


```
my ($matrix_ref); # referinta la tablou de referinte

$matrix_ref =
   $dbh->selectall_arrayref ("SELECT nume, prenume, sufix,"
   . "email, strada, oras, stat, cod_postal, telefon FROM "
   . "membru ORDER BY nume");

# determina dimensiunile matricei
my ($rows) = (!defined ($matrix_ref) ? 0 : scalar (@{$matrix_ref}));
my ($cols) = ($rows == 0 ? 0 : scalar (@{$matrix_ref->[0]}));

for (my $i = 0; $i < $rows; $i++)      # afiseaza fiecare rand
{
my ($delim) = "";
   for (my $j = 0; $j < $cols; $j++)
   {
      print $delim . $matrix_ref->[$i][$j];
      $delim = ",";
   }
   print "\n";
}
```

Verificarea valorilor NULL

Când regăsiți date dintr-o bază de date, poate fi necesar să faceți diferența între valorile coloanelor care sunt `NULL` și valorile zero sau șirurile vide. Acest lucru este ușor de făcut, deoarece `DBI` returnează valorile `NULL` ca `undef`. Totuși, trebuie să fiți sigur că folosiți testul corect. Dacă încercați următorul fragment de program, va afișa `"fals!"` de toate cele trei ori:

```
$col_val = undef; if (!$col_val) { print "fals!\n"; }
$col_val = 0;     if (!$col_val) { print "fals!\n"; }
$col_val = "";    if (!$col_val) { print "fals!\n";}
```

De asemenea, și acest fragment afișează `"fals!"` la ambele teste:

```
$col_val = undef; if ($col_val eq "") { print "fals!\n"; }
$col_val = "";    if ($col_val eq "") { print "fals!\n"; }
```

Acest fragment are același efect:

```
$col_val = "";
if ($col_val eq "")      { print "fals!\n"; }
if ($col_val eq == 0)    { print "fals!\n"; }
```

Pentru a face diferența între valorile `NULL` și valorile diferite de `NULL` din coloane, folosiți `defined()`. După ce ați aflat că o valoare nu reprezintă `NULL`, puteți face diferența între alte tipuri de valori folosind testele adecvate. De exemplu:

```
if (!defined ($col_val)         { print "NULL\n"; }
elsif ($col_val eq "")          { print "sir vid\n"; }
elsif ($col_val == 0)           { print "zero\n"; }
else                            { print "alta valoare\n"; }
```

Este important să efectuați testele într-o ordine corespunzătoare, deoarece atât a doua, cât și a treia comparație sunt adevărate dacă `$col_val` este un șir vid. Dacă inversați ordinea acestor comparații, veți identifica în mod incorect șirurile vide ca fiind zero.

Probleme legate de ghilimele

Până acum, am construit interogări în cel mai elementar mod cu putință, folosind șiruri încadrate între ghilimele simple. Acest fapt creează o problemă la nivelul lexical al limbajului Perl, atunci când șirurile dumneavoastră încadrate între ghilimele conțin valori de asemenea încadrate între ghilimele. De asemenea, puteți avea probleme la nivelul SQL, când doriți să inserați sau să selectați valori care conțin ghilimele, backslash-uri sau date binare. Dacă specificați o interogare sub forma unui șir Perl delimitat prin ghilimele, trebuie să modificați semnificația tuturor aparițiilor caracterului de citare în interiorul șirului de interogare:

```
$query = 'INSERT absente VALUES(14,\'1999-9-16\')';
$query = "INSERT absente VALUES(14,\"1999-9-16\")";
```

Atât Perl, cât și MySQL vă permit să încadrați șiruri între ghilimele, folosind ghilimele simple sau duble, deci uneori puteți evita modificarea semnificației unor caractere prin combinarea semnelor citării:

```
$query = 'INSERT absente VALUES(14,"1999-9-16")';
$query = "INSERT absente VALUES(14,'1999-9-16')";
```

Cu toate acestea, cele două tipuri de ghilimele nu sunt echivalente în Perl. Referințele la variabile sunt interpretate numai în interiorul ghilimelelor duble. Ca atare, ghilimelele simple nu sunt foarte utile atunci când doriți să construiți interogări prin înglobarea referințelor la variabile în șirul de interogare. De exemplu, dacă valoarea variabilei `$var` este `14`, următoarele două șiruri nu sunt echivalente:

```
"SELECT * FROM membru WHERE id = $var"
'SELECT * FROM membru WHERE id = $var'
```

Șirurile sunt interpretate după cum urmează; evident, primul șir este cel care poate fi trimis unui server MySQL:

```
"SELECT * FROM membru WHERE id = 14"
'SELECT * FROM membru WHERE id = $var'
```

O alternativă la încadrarea șirurilor între ghilimele duble constă în utilizarea construcției `qq{}`, care indică programului Perl să trateze întreg textul cuprins între `qq{` și `}` ca pe un șir încadrat între ghilimele duble. (Gândiți-vă că `qq` înseamnă *ghilimele duble*.) De exemplu, următoarele două linii sunt echivalente:

```
$data = "1999-9-16";
$data = qq{1999-9-16};
```

Puteți construi interogări fără a acorda prea mare atenție ghilimelelor dacă folosiți `qq{}`, deoarece puteți folosi ghilimelele (simple *sau* duble) în cadrul șirului de interogare fără a fi necesar să le modificați semnificația. De asemenea, referințele la variabile sunt interpretate. Ambele proprietăți ale construcției `qq{}` sunt prezentate în următoarea interogare:

```
$id = 14;
$data = "1999-9-16";
$interogare = qq{INSERT absente VALUES($id,"$data")};
```


Nu trebuie să folosiți paranteze acolade ca delimitatori pentru construcția `qq`. Alte forme, precum `qq()` și `qq//` sunt de asemenea utilizabile, cu condiția ca delimitatorul de închidere să nu fie conținut în interiorul șirului. Eu prefer `qq{}`, deoarece este mai puțin probabil ca paranteza acoladă să apară în textul șirului interogării decât paranteza rotundă sau caracterul /, după cum este mai puțin probabil să fie confundată cu finalul șirului de interogare decât ultimele două caractere. De exemplu, paranteza rotundă apare în cadrul instrucțiunii `INSERT` prezentate anterior, deci `qq()` nu este o construcție utilă pentru încadrarea între ghilimele a șirului interogării.

Construcția `qq{}` se poate extinde pe mai multe rânduri, ceea ce este util dacă doriți ca șirul interogării să iasă în evidență în raport cu liniile de program Perl înconjurătoare:

```
$id = 14;
$data = "1999-9-16";
$interogare = qq{
   {INSERT absente VALUES($id,"$data")
};
```

Acest lucru este de asemenea util dacă doriți pur și simplu să formatați interogarea pe linii multiple, pentru a o face mai ușor de citit. De exemplu, instrucțiunea `SELECT` din scriptul `dump_members` se prezintă astfel:

```
$sth = $dbh->prepare ("SELECT nume, prenume, sufix, email,"
    . "strada, oras, stat, cod_postal, telefon FROM membru"
    . "ORDER BY nume");
```

Folosind construcția `qq{}`, interogarea poate fi scrisă astfel:

```
$sth = $dbh->prepare (qq{
            SELECT
                    nume, prenume, sufix, email,
                    strada, oras, stat, cod_postal, telefon
            FROM membru
            ORDER BY nume
        });
```

Este la fel de adevărat că șirurile încadrate între ghilimele duble se pot și ele extinde pe mai multe linii. Dar eu prefer mai mult construcția `qq{}` pentru scrierea de șiruri pe linii multiple. Atunci când găsesc într-o linie niște ghilimele duble fără corespondent, prima mea reacție este: „Nu cumva e o eroare de sintaxă?" Apoi, pierd vremea căutând ghilimelele corespondente.

Construcția `qq{}` se ocupă de problemele legate de ghilimele la nivelul lexical al limbajului Perl, astfel încât să puteți insera cu ușurință ghilimele într-un șir, fără ca Perl să protesteze. Totuși, trebuie să vă gândiți și la sintaxa la nivel SQL. Să luăm în considerare această încercare de a insera o înregistrare în tabelul `membru`:

```
$nume = "O'Malley";
$prenume = "Brian";
$data_expirare = "2002-9-1";
$rows = $dbh->do (qq{
```

Continuare

Continuare

```
    INSERT membru (nume,prenume,data_expirare)
    VALUES('$nume','$prenume','$data_expirare')
});
```

Șirul pe care funcția do() îl trimite serverului MySQL se prezintă astfel:

```
INSERT membru (nume,prenume,data_expirare)
VALUES('O'Malley','Brian','2002-9-1')
```

Aceasta nu este o interogare SQL corectă, deoarece un caracter de tip ghilimele simple apare în interiorul unui șir delimitat prin ghilimele simple. Am întâlnit o problemă similară în capitolul 6. Acolo, am rezolvat problema cu ajutorul funcției mysql_escape_string(). DBI furnizează un mecanism similar: pentru fiecare valoare cu ghilimele pe care doriți să o folosiți literal într-o instrucțiune, apelați metoda quote() și folosiți în schimb valoarea returnată de aceasta.

Exemplul anterior poate fi scris într-un mod mai adecvat astfel:

```
$nume = $dbh->quote ("O'Malley");
$prenume = $dbh->quote ("Brian");
$data_expirare = $dbh->quote ("2002-9-1");
$rows = $dbh->do (qq{
    INSERT membru (nume,prenume,data_expirare)
    VALUES($nume,$prenume,$data_expirare)
});
```

Acum, șirul pe care funcția do() îl trimite serverului MySQL se prezintă astfel, ghilimelele care apar în interiorul șirului fiind „camuflate" adecvat pentru server:

```
INSERT membru (nume,prenume,data_expirare)
VALUES('O\'Malley','Brian','2002-9-1')
```

Observați că, atunci când faceți referire la $nume și $prenume în șirul interogării, nu adăugați ghilimele de delimitare; metoda quote() le furnizează automat. Dacă adăugați ghilimele, interogarea dumneavoastră va avea prea multe ghilimele, așa cum se poate vedea în exemplul următor:

```
$value = "paul";
$quoted_value = $dbh->quote ($value);

print " ... WHERE nume = $quoted_value\n";
print " ... WHERE nume = '$quoted_value'\n";
```

Aceste instrucțiuni furnizează următoarele date de ieșire:

```
... WHERE nume = 'paul'
... WHERE nume = ''paul''
```

Câmpuri de înlocuire și asocierea parametrilor

În secțiunile anterioare, am construit interogări prin plasarea valorilor ce urmau a fi inserate în baza de date sau folosite drept criterii de selecție direct în șirul interogării. Acest lucru nu este necesar. DBI vă permite să plasați într-un șir de interogare indicatoare speciale, denumite *câmpuri de înlocuire*, iar apoi să furnizați, la executarea interogării, valorile ce vor fi folosite în locul acestor indicatoare. Principalul motiv pentru această operație îl constituie îmbunătățirea performanțelor, mai ales atunci când executați în mod repetat o interogare, în cadrul unui ciclu.

Ca o ilustrare a modului de lucru a câmpurilor de înlocuire, să presupunem că începeți un nou semestru la școală și doriți să ștergeți tabelul elev pentru catalogul dumneavoastră și apoi să-l inițializați cu noii elevi, folosind o listă a numelor elevilor incluse într-un fișier. În lipsa câmpurilor de înlocuire, puteți șterge conținutul curent al tabelului și încărcați noile nume astfel:

```
$dbh->do (qq{ DELETE FROM elev } );     # sterge randurile existente
while (<>)                              # adauga randuri noi
{
    chomp;
    $_ = $dbh->quote ($_);
    $dbh->do (qq{ INSERT elev SET nume = $_});
}
```

Acest procedeu este ineficient, deoarece forma elementară a interogării INSERT este aceeași de fiecare dată, iar funcția do() apelează prepare() și execute() de fiecare dată pe parcursul ciclului. De asemenea, este mai eficientă apelarea funcției prepare() o singură dată, pentru a configura instrucțiunea INSERT înainte de a introduce ciclul, precum și invocarea funcției execute() o singură dată în cadrul ciclului. Astfel, se evită toate invocările funcției prepare(), cu o singură excepție. DBI ne permite acest lucru, astfel:

```
$dbh->do (qq{ DELETE FROM elev } );    # sterge randurile existente
$sth = $dbh->prepare (qq{ INSERT elev SET nume = ? });
while (<>)                             # adauga randuri noi
{
    chomp;
    $sth->execute ($_);
}
$sth->finish();
```

Observați semnul întrebării din interogarea INSERT. Acela este câmpul de înlocuire. Când este invocată funcția execute(), transferați valoarea care va lua locul câmpului de înlocuire atunci când interogarea este trimisă la server. În general, dacă apelați funcția do() în cadrul unui ciclu, este mai bine să invocați prepare() anterior ciclului, respectiv execute() în interiorul acestuia.

Câteva aspecte de reținut despre câmpurile de înlocuire:

- Nu delimitați caracterul câmp de înlocuire cu ghilimele în interiorul șirului interogării. Dacă o faceți, acesta nu va fi recunoscut drept câmp de înlocuire.
- Nu folosiți metoda quote() pentru a specifica valorile câmpului de înlocuire; în caz contrar, veți obține ghilimele suplimentare în valorile pe care le inserați.
- Într-un șir al unei interogări pot exista mai multe câmpuri de înlocuire, dar nu uitați să transmiteți funcției execute()un număr de valori egal cu indicatoarele pentru câmpurile de înlocuire.

- Fiecare câmp de înlocuire trebuie să specifice o singură valoare, nu o listă de valori. De exemplu, nu puteți prepara și executa o instrucțiune ca aceasta:

```
$sth = $dbh->prepare (qq{{
      INSERT membru nume, prenume VALUES(?)
});
$sth->execute ("Adams,Bill,2003-09-19");
```

Trebuie să procedați astfel:

```
$sth = $dbh->prepare (qq{{
      INSERT membru nume, prenume VALUES(?,?,?)
});
$sth->execute ("Adams","Bill","2003-09-19");
```

- Pentru a specifica `NULL` ca valoare a câmpului de înlocuire, folosiți `undef`.
- Nu încercați să folosiți un câmp de înlocuire pentru cuvintele cheie. Nu veți reuși, deoarece valorile din câmpurile de înlocuire sunt prelucrate automat de funcția `quote()`. Cuvântul cheie va fi plasat în interogare delimitat prin ghilimele, iar interogarea va eșua datorită unei erori de sintaxă.

Pentru unele motoare de baze de date, veți obține o creștere suplimentară a performanței din adăugarea câmpurilor de înlocuire, în afară de o eficiență crescută a ciclurilor. Anumite motoare memorează în cache interogările preparate, precum și planul generat de acestea pentru executarea eficientă a interogărilor. Astfel, dacă aceeași interogare este primită de server mai târziu, va putea fi refolosită fără a se genera un nou plan de execuție. Memorarea în cache a interogărilor este utilă mai ales pentru instrucțiuni `SELECT` complexe, deoarece generarea unui plan bun de execuție poate necesita timp. Câmpurile de înlocuire vă oferă o posibilitate de localizare mai rapidă a interogării în cache, deoarece aceste câmpuri conferă interogărilor un caracter mai generic decât cele construite prin înglobarea anumitor valori din coloane direct în șirul de interogare. Pentru MySQL, câmpurile de înlocuire nu îmbunătățesc performanțele în acest mod, deoarece interogările nu sunt memorate în cache. Totuși, vă puteți scrie propriile interogări folosind câmpurile de înlocuire; dacă se întâmplă să portați un script DBI la un motor care folosește o zonă cache pentru interogări, scriptul dumneavoastră se va executa mai eficient decât în absența câmpurilor de înlocuire.

Câmpurile de înlocuire vă permit să inserați valori într-un șir de interogare în momentul execuției interogării. Cu alte cuvinte, puteți parametriza „datele de intrare" ale interogării. De asemenea, DBI furnizează o operație corespunzătoare la ieșire, denumită *asociere a parametrilor*, care vă permite să parametrizați „datele de ieșire" prin regăsirea automată a valorilor coloanelor în variabile atunci când preluați un rând, fără a trebui să atribuiți personal valori variabilelor.

Să presupunem că aveți o interogare care regăsește numele membrilor din tabelul `membru`. Puteți cere programului DBI să atribuie unor variabile Perl valorile coloanelor selectate. Când preluați un rând, variabilele sunt actualizate automat cu valorile din coloana corespunzătoare. Iată un exemplu care prezintă modul de asociere a coloanelor la variabile și apoi modalitatea de acces la acestea în cadrul ciclului de preluare:


```
$sth = $dbh->prepare (qq{
    SELECT nume, prenume, sufix
    FROM membru
    ORDER BY nume, prenume
});
$sth->execute ();
$sth->bind_col (1, \$nume);
$sth->bind_col (2, \$prenume);
$sth->bind_col (3, \$sufix);
print "$nume, $prenume, $sufix\n" while $sth->fetch;
```

Fiecare apel la funcția `bind_col()` trebuie să specifice un număr de coloană și o referință la variabila pe care doriți să o asociați la coloană. Numerele coloanelor încep de la 1. `bind_col()` trebuie apelată după `execute()`.

Ca alternativă la apelurile individuale la funcția `bind_col()`, puteți transmite toate referințele la variabile într-un singur apel la funcția `bind_columns()`:

```
$sth = $dbh->prepare (qq{
   SELECT nume, prenume, sufix
   FROM membru
   ORDER BY nume, prenume
});
$sth->execute ();
$sth->bind_columns (\$nume, \$prenume, \$sufix);
print "$nume, $prenume, $sufix\n" while $sth->fetch;
```

Specificarea parametrilor de conexiune

Modalitatea cea mai directă de a stabili o conexiune cu serverul este de a specifica toți parametrii de conexiune atunci când invocați metoda `connect()`:

```
$data_source = "DBI:mysql:nume_baza_de_date:nume_gazda";
$dbh->connect ($data_source, nume_utilizator, parola);
```

Dacă omiteți parametrii de conexiune, DBI execută următoarele operații:

- Dacă sursa de date nu este definită sau dacă este șirul vid, se folosește variabila de mediu `DBI_DSN`. Dacă numele de utilizator și parola nu sunt definite (dar nu și dacă acestea sunt alcătuite dintr-un șir vid), se folosesc variabilele de mediu `DBI_USER` și `DBI_PASS`. Sub Windows, dacă numele de utilizator nu este definit, se folosește variabila `USER`.
- Dacă omiteți numele gazdei, aceasta are ca valoare prestabilită `localhost`.
- Dacă specificați `undef` sau un șir vid pentru numele de utilizator, acesta are ca valoare prestabilită numele dumneavoastră UNIX de deschidere a sesiunii de lucru. Sub Windows, numele de utilizator prestabilit este `ODBC`.
- Dacă în locul parolei specificați `undef` sau un șir vid, nu este trimisă nici o parolă.

Puteți specifica anumite opțiuni în sursa de date prin atașarea lor la partea inițială a șirului, fiecare opțiune fiind precedată printr-un caracter punct și virgulă. De exemplu, puteți folosi opțiunea `mysql_read_default_file` pentru a preciza un nume al căii de acces spre un fișier cu opțiuni:

```
$data_source =
    "DBI:mysql:samp_db;mysql_read_default_file=/u/paul/.my.cnf";
```

Când scriptul se execută, va citi fișierul în căutarea parametrilor de conexiune. Să presupunem că `/u/paul/my.cnf` are următorul conținut:

```
[client]
host=pit-viper.snake.net
user=paul
password=secret
```

Apoi, apelul la funcția `connect()` va încerca să se conecteze la serverul MySQL din calculatorul `pit-viper.snake.net` și se va conecta ca utilizatorul `paul` cu parola `secret`. Dacă doriți să permiteți ca scriptul dumneavoastră să fie utilizat de orice persoană care dispune de un fișier cu opțiuni configurat în mod corespunzător, specificați sursa de date astfel:

```
$data_source =
    "DBI:mysql:samp_db;mysql_read_default_file=$ENV{HOME}/.my.cnf";
```

`$ENV{HOME}` conține calea de acces spre catalogul de bază al utilizatorului care rulează scriptul, deci numele gazdei, numele utilizatorului și parola pe care le folosește scriptul vor fi extrase din fișierul cu opțiuni al fiecărui utilizator. Scriind un script în acest mod, nu trebuie să înglobați literalmente parametrii de conexiune în script.

De asemenea, puteți folosi opțiunea `mysql_read_default_group`, pentru a specifica un grup din fișierul cu opțiuni. Aceasta determină în mod automat citirea fișierului `.my.cnf` al utilizatorului și vă permite să specificați un grup de opțiuni care va fi citit alături de grupul `[client]`. De exemplu, dacă aveți opțiuni care sunt specifice scripturilor dumneavoastră DBI, le puteți menționa într-un grup `[dbi]`, după care folosiți o valoare a sursei de date ca aceasta:

```
$data_source =
    "DBI:mysql:samp_db;mysql_read_default_group=dbi";
```

mysql_read_default_file și `mysql_read_default_group` necesită MySQL 3.22.10 sau o versiune mai recentă, precum și DBD::mysql 1.21.06 sau mai recent. Pentru mai multe detalii privind opțiunile pentru specificarea șirului sursă de date, vezi Anexa G. Pentru mai multe informații despre formatul fișierelor cu opțiuni MySQL, vezi Anexa E, „Referință de program MySQL".

Utilizarea unui fișier cu opțiuni nu vă împiedică să specificați parametrii de conexiune în apelul la funcția `connect()` (de exemplu, dacă doriți ca scriptul să se conecteze ca un anumit utilizator). Orice valoare explicită a numelui gazdei, a numelui de utilizator sau a parolei specificate în apelul la funcția `connect()` va redefini parametrii de conexiune aflați în fișierul cu opțiuni. De exemplu, puteți dori ca scriptul dumneavoastră să analizeze opțiunile `--host`, `--user` și `--password` din linia de comandă și să folosească aceste valori, dacă sunt date, cu prioritate față de orice valoare din fișierul cu opțiuni. Acest fapt este util, deoarece este modalitatea standard de comportare a clienților MySQL. Ca atare, scripturile dumneavoastră DBI vor manifesta aceeași comportare.

Pentru celelalte scripturi în linie de comandă pe care le vom crea în acest capitol, voi folosi linii de program standard pentru stabilirea, respectiv întreruperea conexiunii. Le voi prezenta o singură dată aici, astfel încât să ne putem concentra asupra corpului principal al fiecărui script atunci când îl vom scrie:


```
#! /usr/bin/perl

use DBI;
use strict;

# analizeaza parametrii de conexiune din linia de comanda,
# daca acestia sunt dati

use Getopt::Long;
$Getopt::Long::ignorecase = 0; # optiunile sunt sensibile la

# diferenta intre majuscule si minuscule
# toti parametrii prestabiliti lipsesc
my ($host_name, $user_name, $password) = (undef, undef, undef);

# GetOptions nu pare sa permita forma —uuser_name,
# numai —u user_name?
GetOptions(
   # =s inseamna ca dupa optiune este necesar un argument sir
   "host/h=s"              => \$host_name
   ,"user/u=s"             => \$user_name
   # :s inseamna ca dupa optiune argumentul sir este optional
   ,"password/p:s"         => \$password

) or exit (1);

# solicita parola daca optiunea este specificata fara valoare
if (defined ($password) && !$password)
{
   # dezactiveaza reflectarea, dar nu interfereaza cu STDIN
   open (TTY, "/dev/tty") or die ("Nu poate deschide terminalul\n");
   system ("stty -echo < /dev/tty/");
   print STDERR "Introduceti parola: ";
   chomp ($password = <TTY>);
   system ("stty echo < /dev/tty");
   close (TTY);
   print STDERR "\n";
}

# construieste sursa de date

my ($dsn) = "DBI:mysql:samp_db";
```

Continuare

Continuare

```
$dsn .= ":hostname=$host_name" if $host_name;
$dsn .= ";mysql_read_default_file=$ENV{HOME}/.my.cnf";

# conectare la server
my (%attr) = ( RaiseError => 1 );
my ($dbh) = DBI->connect ($dsn, $user_name, $password, \%attr);
```

Acest program inițializează DBI, caută parametrii de conexiune în linia de comandă și apoi stabilește conexiunea cu serverul MySQL folosind parametri din linia de comandă sau parametri din fișierul ~/my.cnf al utilizatorului care rulează scriptul. În cazul în care configurați fișierul my.cnf în catalogul dumneavoastră de bază, nu va trebui să introduceți nici un parametru de conexiune atunci când rulați scripturile. (Nu uitați să configurați modul astfel încât nimeni altcineva să nu mai poată citi fișierul. Pentru instrucțiuni, consultați Anexa E.)

Partea finală a scripturilor noastre va fi de asemenea identică pentru toate scripturile; pur și simplu se suspendă conexiunea și se încheie execuția programului:

```
$dbh->disconnect ();
exit (0);
```

Când vom ajunge la secțiunea de programare Web, „Utilizarea DBI în aplicații Web", vom modifica puțin programul de configurare a conexiunii, dar ideea de bază va rămâne aceeași.

Depanare

Când doriți să depanați un script DBI care nu funcționează adecvat, de obicei se folosesc două tehnici, fie separat, fie în tandem. Mai întâi, puteți „împrăștia" instrucțiuni de afișare pe tot cuprinsul scriptului dumneavoastră. Astfel, vă puteți ajusta datele de ieșire ale procesului de depanare conform preferințelor, dar trebuie să adăugați instrucțiunile manual. În al doilea rând, puteți folosi funcționalitățile de urmărire ale programului DBI. Aceasta este o opțiune mai generală, dar mai sistematică, și care intră în acțiune imediat după ce ați activat-o. De asemenea, vă prezintă informații despre funcționarea driverului pe care nu le puteți obține altfel.

Depanarea folosind instrucțiuni de afișare

O întrebare frecventă în lista de corespondență pentru MySQL este următoarea: „Am o interogare care funcționează bine când o execut în mysql, dar care nu funcționează din scriptul DBI. Cum vine asta?" Este ceva obișnuit ca un script DBI să emită o altă interogare decât cea dorită de operator. Dacă afișați o interogare înainte de a o executa, poate veți fi surprins să vedeți ceea ce trimiteți efectiv serverului. Să presupunem că o interogare pe care o tastați se prezintă astfel (fără caracterul punct și virgulă de terminare):

```
INSERT membru (nume,prenume,data_expirare)
VALUES("Vasile","Cornel","2002-6-3")
```

Apoi, încercați același lucru într-un script DBI:


```
$nume = "Vasile";
$prenume = "Cornel";
$data_expirare = "2002-6-3";
$query =
qq{
   INSERT membru (nume,prenume,data_expirare)
   VALUES($nume,$prenume,$data_expirare)
};
$rows =dbh->do ($query);
```

Interogarea nu va funcționa, chiar dacă este aceeași cu prima. Dar este oare? Încercați să o afișați:

```
print "$query\n";
Iata rezultatul:
INSERT membru (nume,prenume,data_expirare)
VALUES(Vasile,Cornel,2002-6-3)
```

Din aceste date de ieșire, puteți observa că ați uitat să scrieți ghilimelele în jurul valorilor din coloane în lista VALUES(). Modalitatea corectă de specificare a interogării este aceasta:

```
$nume = $dbh->quote ("Vasile");
$prenume = $dbh->quote ("Cornel");
$data_expirare = $dbh->quote ("2002-6-3");
$query =
qq{
   INSERT membru (nume,prenume,data_expirare)
   VALUES($nume,$prenume,$data_expirare)
};
```

Alternativ, vă puteți specifica interogarea folosind câmpuri de înlocuire și transferați direct metodei do()valorile ce urmează a fi inserate:

```
$nume = "Vasile";
$prenume = "Cornel";
$data_expirare = "2002-6-3";
$query =
qq{
   INSERT membru (nume,prenume,data_expirare)
   VALUES(?,?,?)
};
$rows =dbh->do ($query, undef, $nume, $prenume, $data_expirare);
```

Din păcate, atunci când procedați astfel, nu puteți vedea care este aspectul interogării complete folosind o instrucțiune de afișare, deoarece valorile din câmpurile de înlocuire sunt evaluate numai când invocați metoda do(). Când folosiți câmpuri de înlocuire, urmărirea poate fi o metodă de depanare mai utilă.

Depanarea prin urmărire

Puteți cere programului DBI să genereze informații de urmărire (depanare) atunci când încercați să determinați motivul pentru care un script nu funcționează corect. Nivelurile de urmărire variază de la 0 (inexistent) la 9 (aport maxim de informații). În general, nivelurile de urmărire 1 și 2 sunt cele mai utile. O urmărire de nivel 2 vă prezintă textele interogărilor pe care le executați (inclusiv rezultatul inserției valorilor în câmpurile de înlocuire), rezultatul apelurilor la funcția `quote()` etc. Aceasta poate fi de mare ajutor în depistarea unei probleme.

Puteți controla nivelul de urmărire din interiorul scripturilor individuale folosind metoda `trace()` sau puteți configura variabila de mediu `DBI_TRACE`, pentru a influența urmărirea tuturor scripturilor DBI pe care le rulați.

Pentru a folosi apelul la funcția `trace()`, transferați un argument nivel de urmărire și, opțional, un nume de fișier. Dacă nu specificați nici un nume de fișier, toate datele de ieșire ale procesului de urmărire ajung la `STDERR`; în caz contrar, ajung la fișierul denumit. Câteva exemple:

`DBI->trace (1)`	Urmărire de nivel 1 cu ieșire în STDERR
`DBI->trace (2, "trace.out")`	Urmărire de nivel 2 cu ieșire în „trace.out“
`DBI->trace (0)`	Dezactivarea datelor de ieșire ale urmăririi

Când funcția este invocată sub forma `DBI->trace()`, sunt urmărite toate operațiile DBI. Pentru o abordare mai detaliată, puteți activa urmărirea la nivelul variabilei de manipulare individuale. Acest lucru este util când știți bine unde se află o anumită problemă în scriptul dumneavoastră și nu doriți să parcurgeți datele de ieșire ale procesului de urmărire, pentru a afla tot ceea ce s-a produs până în momentul respectiv. De exemplu, dacă aveți probleme cu o anumită interogare `SELECT`, puteți urmări variabila de manipulare pentru instrucțiuni asociată interogării:

```
$sth-> $dbh->prepare (qq{ SELECT ... });    # creeaza variabila de
                                            # manipulare a instructiunii
$sth->trace (1);                            # activeaza urmarirea
                                            # instructiunii
$sth->execute ();
```

Dacă specificați un argument nume de fișier în orice apel la funcția `trace()`, indiferent dacă pentru întreg scriptul DBI sau pentru o variabilă de manipulare individuală, *toate* datele de ieșire ale procesului de urmărire ajung la fișierul respectiv.

În vederea activării urmăririi la nivel global pentru toate scripturile DBI pe care le rulați, configurați variabila de mediu `DBI_TRACE` din interpretorul dumneavoastră. Sintaxa necesară depinde de interpretorul pe care îl folosiți:

`% setenv DBI_TRACE` *valoare*	Pentru `csh`, `tcsh`
`$ DBI_TRACE=`*valoare*	Pentru `sh`, `ksh`, `bash`
`$ export DBI_TRACE`	
`C:\> set DBI_TRACE=`*valoare*	Pentru Windows

Formatul pentru *valoare* este același pentru toate interpretoarele: un număr `n` pentru activarea urmăririi la nivelul `n` cu ieșire în `STDERR`; un nume de fișier pentru activarea urmăririi de nivel 2 cu ieșire în fișierul denumit, respectiv `n=nume_fisier` pentru activarea urmăririi de nivel `n` cu ieșire în fișierul denumit. Iată câteva exemple care folosesc sintaxa `csh`:

`% setenv DBI_TRACE 1`	Urmărire de nivel 1 cu ieșire în `STDERR`
`% setenv DBI_TRACE 1=trace.out`	Urmărire de nivel 1 în „trace.out“
`% setenv DBI_TRACE trace.out`	Urmărire de nivel 2 în „trace.out“

Dacă activați urmărirea într-un fișier din interpretor, nu uitați să o dezactivați după ce ați rezolvat problema. Datele de ieșire de la depanare sunt anexate la fișierul de urmărire fără a-l suprascrie, deci fișierul poate deveni foarte mare dacă nu sunteți atent. Este total contraindicat să definiți `DBI_TRACE` într-un fișier de pornire din interpretor, precum `.cshrc`, `.login` sau `.profile`! Sub UNIX, puteți dezactiva urmărirea folosind oricare din următoarele comenzi (sintaxă `csh`):

```
% setenv DBI_TRACE 0
% unsetenv DBI_TRACE
```

Pentru `sh`, `ksh` sau `bash` procedați astfel:

```
$ DBI_TRACE=0
$ export DBI_TRACE
```

În Windows, puteți dezactiva caracteristica de urmărire folosind oricare din aceste programe:

```
C:\> unset DBI_TRACE
C:\> set DBI_TRACE=0
```

Utilizarea metadatelor aferente setului de rezultate

Puteți folosi DBI pentru a obține accesul la metadatele setului de rezultate, adică la informații descriptive despre rândurile selectate de o interogare. Pentru a obține aceste informații, obțineți accesul la atributele variabilei de manipulare pentru instrucțiuni asociată cu interogarea care a generat setul de rezultate. Unele din aceste atribute sunt furnizate ca atribute DBI standard care sunt disponibile pentru toate driverele de baze de date (precum `NUM_OF_FIELDS`, numărul coloanelor din setul de rezultate). Altele, specifice sistemului MySQL, sunt furnizate de DBD::mysql, driverul MySQL pentru DBI. Aceste atribute, precum `mysql_max_length`, care indică lățimea maximă a valorilor din fiecare coloană, nu sunt aplicabile altor motoare de baze de date. În măsura în care folosiți oricare dintre atributele specifice MySQL, riscați ca scripturile dumneavoastră să devină neportabile pe alte baze de date. Pe de altă parte, pot facilita obținerea informațiilor pe care le doriți.

Trebuie să solicitați metadatele la momentul adecvat. În general, atributele setului de rezultate nu sunt disponibile pentru o instrucțiune `SELECT` decât după ce ați invocat `prepare()` și `execute()`. În plus, atributele pot deveni invalide după ce invocați `finish()`.

Să vedem cum se poate folosi unul din atributele metadatelor MySQL, și anume `mysql_max_length`, în conjuncție cu atributul de nivel DBI `NAME`, care conține numele coloanelor din interogare. Putem combina informațiile furnizate de aceste atribute pentru a scrie un script `box_out`, care generează date de ieșire ale interogărilor `SELECT` în

același stil „încasetat" pe care îl obțineți când rulați programul client `mysql` în mod interactiv. Corpul principal al scriptului `box_out` este următorul (puteți înlocui instrucțiunea `SELECT` cu oricare alta; rutinele de scriere a datelor de ieșire sunt independente de interogarea în sine):

```
my ($sth) = $dbh->prepare (qq{
   SELECT nume, prenume, oras, stat
   FROM presedinte ORDER BY nume, prenume
});
$sth->execute ();              # atributele trebuie sa fie disponibile
                               # dupa acest apel

# latimile maxime efective ale valorilor coloanelor din
# setul de rezultate
my (@wid) = @{$sth->{mysql_max_length}};
# numar de coloane
my ($ncols) = scalar (@wid);

# ajusteaza latimile coloanelor in cazul in care antetele
# coloanelor sunt mai late decat valorile datelor
for (my $i = 0; $i < $ncols; $i++)
{
my ($name_wid) = length ($sth->{NAME}->[$i]);

   $wid[$i] = $name_wid if $wid[$i] < $name_wid;
}

# afiseaza datele de iesire
print_dashes (\@wid, $ncols);                # rand de liniute
print_row ($sth->{NAME}, \@wid, $ncols);     # antete de coloana
print_dashes (\@wid, $ncols);                # rand de liniute
while (my $ary_ref = $sth->fetch)
{
   print_row ($ary_ref, \@wid, $ncols);      # valori date din rand
}
print_dashes (\@wid, $ncols);                # rand de liniute
$sth->finish();
```

După ce interogarea a fost inițiată cu `execute()`, putem prelua metadatele de care avem nevoie. `$sth->{NAME}` și `$sth->{mysql_max_length}` oferă numele coloanelor și lățimea maximă a valorilor din fiecare coloană. Valoarea fiecărui atribut este o referință la un tablou care conține câte o valoare pentru fiecare coloană a setului de rezultate, în ordinea în care sunt specificate coloanele în interogare.

Calculele rămase sunt foarte asemănătoare celor folosite pentru programul `client5` prezentat în capitolul 6. De exemplu, pentru a preveni alinierea eronată a datelor de ieșire, vom ajusta lățimile coloanelor de jos în sus, dacă numele unei coloane este mai lat decât oricare din valorile datelor din coloană.

Funcțiile de ieșire, `print_dashes()` și `print_row()`, sunt scrise după cum urmează. Și acestea sunt similare programului corespunzător din `client5`:

```
sub print_dashes
{
my ($wid_ary_ref) = shift;        # latimile coloanelor
my ($cols) = shift;               # numarul de coloane

   print "+";
   for (my $i = 0; $i < $cols; $i++)
   {
      print "-" x ($wid_ary_ref->[$i]+2) . "+";
   }
   print "\n";
}

# afiseaza un rand de date.
# (nu aliniaza la dreapta coloanele numerice)

sub print_row
{
my ($val_ary_ref) = shift;       # valorile din coloane
my ($wid_ary_ref) = shift;       # latimile coloanelor
my ($cols) = shift;              # numarul de coloane

   print "|";
   for (my $i = 0; $i  $cols; $i++)
   {
      printf " %-*s |", $wid_ary_ref->[$i],
            defined ($val_ary_ref->[$i] ? $val_ary_ref->[$i] : " NULL ";
   }
   print "\n";
}
```

Datele de ieșire ale scriptului `box_out` sunt următoarele:

nume	prenume	oras	stat
Adams	John	Braintree	MA
Adams	John Quincy	Braintree	MA
Arthur	Chester A.	Fairfield	VT
Buchanan	James	Mercersburg	PA
Bush	George W.	Milton	MA

Următorul nostru script folosește metadatele referitoare la coloane pentru a produce date de ieșire într-un alt format. Acest script, `show_member`, vă permite să examinați rapid datele privind membrii Ligii istorice, fără a introduce nici o interogare. Dacă se dă numele unui membru, intrarea selectată este afișată astfel:

```
% show_member artel
nume:            Artel
prenume:         Mike
sufix:
data_expirare:   2003-04-16
email:           mike_artel@venus.org
strada:          4264 Lovering Rd.
oras:            Miami
stat:            FL
cod_postal:      12777
telefon:         075-961-0712
interese:        drepturile civile,invatamant,razboiul revolutionar
membru_id:       63
```

De asemenea, puteți invoca `show_members` folosind identificatorul de membru sau utilizând un model care să corespundă mai multor nume de familie. Comenzile următoare afișează intrarea pentru numărul 23 și intrările pentru membrii ale căror nume de familie începe cu litera C:

```
% show_member 23
% show_member c%
```

Corpul principal al scriptului `show_member` este prezentat mai jos. Scriptul folosește atributul `NAME` pentru a determina etichetele de utilizat pentru fiecare rând al datelor de ieșire, respectiv atributul `NUM_OF_FIELDS` pentru a afla numărul de coloane pe care îl conține setul de rezultate:

```
my ($count) = 0;    # numarul de intrari afisate pana acum
my ($label) = ();   # tabloul cu etichetele de coloana
my ($label_wid) = 0;

while (@ARGV)       # ruleaza interogarea pentru fiecare argument
                    # din linia de comanda
{
    my ($arg) = shift (@ARGV);
    my ($sth, $clause, $address);

    # in mod prestabilit se cauta dupa nume; cauta dupa
    # identificator daca argumentul este numeric
    $clause = "nume LIKE " . $dbh->quote ($arg);
    $clause = "membru_id = " . $dbh->quote ($arg) if $arg =~ /^\d/;

    # emite interogarea
```



```
    $sth = $dbh->prepare (qq{
        SELECT * FROM membru
        WHERE $clause
        ORDER BY nume, prenume
    });
    $sth->execute ();

    # obtine numele coloanelor de utilizat in etichete si determina
    # latimea maxima a numelor coloanelor pentru formatare (faceti
    # aceasta numai la prima parcurgere a ciclului)
    if ($label_wid == 0)
    {
        @label = @{$sth->{NAME}};
        foreach my $label (@label)
        {
            $label_wid = length ($label) if $label_wid < length ($label)
        }
    }

    # citeste si afiseaza rezultatele interogarii, apoi face curatenie
    while (my @ary = $sth-fetchrow_array ())
    {
        # afiseaza linie noua inainte de intrarea a doua
        # si de intrarile ulterioare
        print "\n" if ++$count > 1;
        foreach (my $i = 0; $i < $sth->{NUM_OF_FIELDS}; $i++)
        {
            # afiseaza eticheta si valoarea, daca exista
            printf "%-*s", $label_wid+1, $label[$i] . ":";
            print " " . $ary[$i] if $ary[$i];
            print "\n";
        }
    }
    $sth->finish ();
}
```

Rolul scriptului `show_member` este de a prezenta întregul conținut al unei intrări, indiferent care sunt câmpurile. Folosind `SELECT *` pentru a regăsi toate coloanele și atributul `NAME` pentru a afla care sunt acestea, scriptul respectiv va funcționa fără modificări, chiar dacă sunt adăugate sau eliminate coloane din tabelul `membru`.

Dacă nu doriți decât să aflați care sunt coloanele pe care le conține un tabel, fără a regăsi vreun rând, puteți emite această interogare:

```
SELECT * FROM nume_tabel WHERE 1 = 0
```

După ce invocați `prepare()` și `execute()` ca de obicei, puteți obține numele coloanelor din `@{$sth->{NAME}}`. Rețineți, totuși, că deși acest mic truc de utilizare a unei interogări „vide" funcționează pentru MySQL, nu este portabil și nu se poate aplica tuturor motoarelor de baze de date.

Pentru mai multe informații privind atributele furnizate de DBI și de DBD::mysql, consultați Anexa G. Este la latitudinea dumneavoastră să determinați dacă doriți să obțineți un maxim de portabilitate evitând atributele specifice sistemului MySQL sau să beneficiați de ele, cu prețul reducerii portabilității.

Utilizarea DBI

Până acum ați văzut un număr de concepte implicate în programarea DBI, deci să trecem la unele din lucrurile pe care doream să le putem face în ceea ce privește baza noastră de date demonstrativă. Scopurile noastre au fost schițate la început în capitolul 1, „Introducere în MySQL și SQL". Cele care vor fi abordate prin scrierea scripturilor DBI din capitolul de față sunt enumerate aici.

Pentru proiectul de evidență a rezultatelor școlare, dorim să putem regăsi punctaje pentru orice chestionar sau test dat.

Pentru Liga istorică, dorim să putem executa următoarele operații:

- Generarea catalogului membrilor în diferite formate. Dorim o listă numai cu nume, care va fi utilizată în programul editat cu ocazia banchetului anual, într-un format care se poate folosi pentru generarea catalogului tipărit.
- Identificarea acelor membri ai Ligii care trebuie să-și plătească în curând cotizația, urmată de trimiterea unui mesaj e-mail pe adresa acestora, pentru a-i anunța.
- Editarea intrărilor aferente membrilor. (La urma urmelor, va trebui să le actualizăm datele de expirare după ce aceștia își achită cotizațiile.)
- Identificarea membrilor cu interese comune.
- Publicarea catalogului pe Internet.

Pentru unele din aceste sarcini, vom scrie scripturi care rulează de la linia de comandă. Pentru celelalte, vom crea scripturi în secțiunea următoare, „Utilizarea DBI în aplicațiile Web", pe care le puteți folosi în conjuncție cu serverul dumneavoastră de Web. La sfârșitul capitolului, vom mai rămâne cu un număr de sarcini neîndeplinite. Le vom rezolva și pe acestea în capitolul 8, „Interfața API pentru PHP".

Generarea catalogului Ligii istorice

Unul dintre scopurile noastre este de a genera informații din catalogul Ligii istorice în diferite formate. Cel mai simplu format pe care îl vom genera este o listă cu numele membrilor pentru programul banchetului anual. Aceasta poate fi un listing în format text simplu. Lista va deveni o parte a documentului mai mare folosit pentru a crea programul, deci tot ceea ne trebuie este o entitate care poate fi „lipită" în documentul respectiv.

Pentru catalogul în format acceptabil pentru tipar, este necesară o reprezentare mai adecvată decât aceea în format text simplu, deoarece dorim un text cu o formatare estetică. O opțiune rezonabilă aici este RTF (Rich Text Format), un format creat de Microsoft și

acceptat de numeroase procesoare de text. Word este un asemenea program, desigur, dar RTF este acceptat și de multe alte programe, precum WordPerfect și AppleWorks. Diferitele procesoare de text acceptă RTF în moduri diferite, dar vom folosi un subset elementar al specificației RTF complete, care trebuie să fie acceptat de orice procesor de texte care prezintă chiar și cel mai redus nivel posibil de compatibilitate cu RTF.

Procedurile pentru generarea formatelor de catalog de tip listă pentru banchet, respectiv RTF, sunt în esență aceleași: emiteți o interogare pentru regăsirea intrărilor și apoi rulați un ciclu care preia și formatează fiecare intrare. Datorită acestei asemănări fundamentale, ar fi interesant să evităm scrierea a două scripturi diferite. În acest sens, să scriem un singur script, intitulat `gen_cat`, care poate genera date de ieșire din catalog în diferite formate. Putem structura scriptul după cum urmează:

1. Înainte de a scrie conținutul intrărilor, efectuați toate inițializările necesare pentru formatul datelor de ieșire. Pentru lista membrilor care se va insera în programul pentru banchet nu este necesară nici o inițializare specială, dar va trebui să scriem un limbaj de control inițial pentru versiunea RTF.
2. Preluați și afișați fiecare intrare, formatată adecvat pentru tipul datelor de ieșire dorit.
3. După ce toate intrările au fost prelucrate, efectuați toate operațiile necesare de curățenie și de terminare a programului. Din nou, pentru lista de la banchet nu este nevoie de operații de manipulare speciale, dar pentru versiunea RTF este necesar un limbaj de control al închiderii.

În viitor, vom folosi acest script pentru a scrie date de ieșire în alte formate, deci îl vom face extensibil prin configurarea unei „cutii de distribuție" – o combinație cu un element pentru fiecare format al datelor de ieșire. Fiecare element specifică funcții care generează date de ieșire în mod adecvat pentru un format dat: o funcție de inițializare, o funcție de scriere a intrărilor și o funcție de curățenie:

```
# cutie de distributie care contine functii de formatare
# pentru fiecare format al datelor de iesire
my (%cutie_distributie) =
  (
    "banquet" =>                  # functii pentru lista de banchet
    {
        "init"   => undef,        # nu este necesara nici
                                  # o initializare
        "entry"   => \&format_banquet_entry,
        "cleanup" => undef           # curatenia nu este necesara
    },
    "rtf" =>                         # functii pentru formatul RTF
    {
        "init"    => \&rtf_init,
        "entry"   => \&format_rtf_entry,
        "cleanup" => \&rtf_cleanup
    }
  );
```

Fiecare element din cutia de distribuție include un nume de format drept cheie (în cazul nostru „banquet" și „rtf"). Vom scrie scriptul astfel încât dumneavoastră să specificați numai formatul dorit în linia de comandă atunci când îl rulați:

```
% gen_cat banquet
% gen_cat rtf
```

Prin configurarea unei „cutii de distribuție" în acest mod, putem adăuga cu ușurință funcționalitatea unui nou format:

1. Scrieți trei funcții de formatare.
2. Adăugați la cutia de distribuție un element nou, care indică spre aceste funcții.
3. Pentru a produce date de ieșire în noul format, invocați `gen_cat` și specificați numele formatului în linia de comandă.

Programul pentru selectarea intrării adecvate din „cutia de distribuție" în conformitate cu primul argument din linia de comandă este prezentat mai jos și se bazează pe faptul că numele formatelor datelor de ieșire reprezintă cheile din hash-ul `%cutie_distributie`. Dacă în cutia de distribuție nu exista o asemenea cheie, formatul este incorect. Nu este necesară codarea hard a numelor formatelor în program; dacă la cutia de distribuție este adăugat un format nou, acesta este detectat automat. Dacă în linia de comandă nu este specificat nici un nume de format sau este precizat un format incorect, scriptul produce un mesaj de eroare și afișează o listă a numelor permise:

```
# verifica daca a fost specificat un argument in linia de comanda
@ARGV = = 1
   or die "Utilizare : gen_cat tip_format\nFormate permise:"
      . join (" ", sort (keys (%cutie_distributie))) . "\n";

# determina intrarea adecvata din cutia de distributie din
# argumentul liniei de comanda; daca nu este gasita nici o
# intrare, tipul de format a fost incorect
my ($func_hashref) = $cutie_distributie{$ARGV[0]};

defined ($func_hashref)
   or die "Format necunoscut: $ARGV[0]\nFormate permise: "
      . join (" ", sort (keys (%cutie_distributie))) . "\n";
```

Dacă în linia de comandă este specificat un nume de format corect, programul precedent configurează funcția `$func_hashref`. Valoarea sa va fi o referință spre hash, care indică spre funcțiile de scriere a datelor de ieșire pentru formatul selectat. Apoi, putem rula interogarea de selecție a intrării. Apoi, invocăm funcția de inițializare, preluăm și afișăm intrările și invocăm funcția de curățenie:

```
# emite interogarea
my ($sth) = $dbh->prepare (qq{
   SELECT * FROM membru ORDER BY nume, prenume
});
$sth->execute ();
```



```
#invoca functia de initializare, daca exista vreuna
&{$func_hashref->{init}} if defined ($func_hashref->{init});

# preia si afiseaza intrarile, daca exista vreo functie de
# formatare de datelor de intrare
if (defined ($func_hashref->{entry}))
{
   while (my $entry_ref = $sth->fetchrow_hashref ("NAME_lc"))
   {
      # transmite referinta la intrare functiei de formatare
      &{$func_hashref->{entry}} ($entry_ref);
   }
}
$sth->finish ();

# invoca functia de curatenie, daca exista vreuna
&{$func_hashref->{cleanup}} if defined ($func_hashref->{cleanup});
```

Ciclul de preluare a intrărilor folosește `fetchrow_hashref()` dintr-un anumit motiv. Dacă ciclul prelua un tablou, funcțiile de formatare ar fi trebuit să cunoască ordinea coloanelor. O puteți determina prin accesul la atributul `$sth->{NAME}` (care conține numele coloanelor în ordinea în care au fost returnate), dar de ce să vă obosiți? Folosind o referință la hash, funcțiile de formatare pot denumi exact coloanele de care au nevoie folosind construcția `$entry_ref->(nume_coloana)`. Acest procedeu nu este deosebit de eficient, dar este ușor și poate fi folosit pentru orice format pe care dorim să-l generăm, deoarece știm că toate câmpurile de care avem nevoie se găsesc în hash.

Tot ce ne rămâne de făcut este să scriem funcțiile pentru fiecare format al datelor de ieșire (adică pentru funcțiile denumite de intrările din cutia de distribuție).

Generarea listei cu membri pentru programul banchetului anual

Pentru acest format al datelor de ieșire, avem nevoie pur și simplu de numele membrilor. Nu sunt necesare apeluri la funcții de inițializare sau curățenie. Ne trebuie numai o funcție de formatare a intrării:

```
sub format_banquet_entry
{
   my ($entry_ref) = shift;
   my ($nume);

   $nume = $entry_ref->{prenume} . " " . $entry_ref->{nume};
   if ($entry_ref->{sufix})       # numele are un sufix
   {
      # fara virgule pentru sufixele I, II, III etc.
      $nume .= "," unless $entry_ref->{sufix} =~ /^[IVX]+$/;
```

Continuare

Continuare

```
        $nume .= " " . $entry_ref->{sufix} if $entry_ref->{sufix};
    }
    print "$nume\n";
}
```

Argumentul funcției `format_banquet_entry()` este o referință la hash-ul valorilor din coloane aferente unui rând. Funcția alătură numele și prenumele, plus un eventual sufix. Poanta este că sufixe precum „Jr." sau „Sr." trebuie precedate de o virgulă și de un spațiu, în timp ce sufixele de forma „II" sau „III" trebuie să fie precedate numai de un spațiu:

```
Michael Alvis IV
Clarence Elgar, Jr.
Bill Matthews, Sr.
Mark York II
```

Deoarece literele I, V și X acoperă toate generațiile, de la prima la a 39-a[1], putem determina dacă vom insera sau nu o virgulă folosind următorul test:

```
$nume .= "," unless $hash_ref->{sufix} =~ /^[IVX]+$/;
```

Programul aferent funcției `format_banquet_entry()` care alătură numele și prenumele ne va fi necesar și pentru versiunea RTF a catalogului. Totuși, în loc de a duplica programul respectiv în funcția `format_rtf_entry()`, îl vom „înghesui" într-o funcție:

```
sub format_name
{
my ($entry_ref) = shift;
my ($nume);

    $nume = $entry_ref->{prenume} . " " . $entry_ref->{nume};
    if ($entry_ref->{sufix})          # numele are un sufix
    {
       # fara virgule pentru sufixele I, II, III etc.
       $nume .= "," unless $entry_ref->{sufix} =~ /^[IVX]+$/;
       $nume .= " " . $entry_ref->{sufix} if $entry_ref->{sufix};
    }
    return "$nume";
}
```

Prin inserția în funcția `format_name()` a programului care determină șirul numelui, funcția `format_banquet_entry()` se reduce aproape la zero:

```
sub format_banquet_entry
{
    printf "%s\n", format_name ($_[0]);
}
```

[1] Pentru că echivalentul în cifre romane al numărului 40 este XL. – N.T.

Generarea catalogului în format tipărit

Generarea versiunii RTF a catalogului este puțin mai complicată decât generarea listei cu membri pentru programul editat cu ocazia banchetului. Un motiv ar fi acela că trebuie să afișăm mai multe informații din fiecare intrare. Un al doilea motiv: este necesară inserția unui limbaj de control RTF la fiecare intrare, pentru a obține efectele dorite. Un cadru minimal pentru un document RTF se prezintă astfel:

```
{\rtf0
{\fonttbl {\f0 Times;}}
\plain \f0 \fs24
   ...continutul documentului se insereaza aici...
}
```

Documentul începe și se termină cu paranteze acolade. Cuvintele cheie RTF încep cu un backslash, iar primul cuvânt cheie al documentului trebuie să fie `\rtfn`, unde `n` este numărul de versiune al specificației RTF căreia îi corespunde documentul. Pentru exemplul nostru, versiunea 0 este suficientă.

În cadrul documentului, specificăm o tabelă de fonturi pentru a indica fontul care se va folosi la intrări. Informațiile din tabela cu fonturi sunt prezentate într-un grup alcătuit din paranteze acolade, care includ un cuvânt cheie inițial `\fonttbl` și unele informații despre fonturi. Tabela de fonturi prezentată în cadru definește numărul de font 0 ca fiind în Times. (Avem nevoie de un singur font, dar puteți folosi mai multe dacă doriți ca documentul să aibă un aspect mai estetic.)

Următoarele câteva directive configurează stilul de formatare prestabilit: `\plain` selectează formatul simplu, `\f0` selectează fontul 0 (pe care l-am definit ca Times în tabela cu fonturi), iar `\fs24` configurează dimensiunea fontului la 12 puncte (numărul de după `fs` indică dimensiunea în jumătăți de punct). Nu este necesar să configurați margini de pagină; majoritatea procesoarelor de texte vor furniza valori prestabilite rezonabile.

Pentru a alege o abordare foarte simplă, putem afișa fiecare intrare ca o serie de linii, cu o etichetă pe fiecare linie. Dacă informațiile corespunzătoare unei anumite linii din datele de ieșire lipsesc, linia este omisă. (De exemplu, linia „Email:" nu este afișată pentru membrii care nu dispun de adresă de e-mail.) Unele linii (precum linia „Adresa:") sunt alcătuite din informații plasate în mai multe coloane (stradă, oraș, stat, cod poștal), astfel încât scriptul să poată manevra mai multe combinații de valori care lipsesc. Iată un exemplu al formatului datelor de ieșire pe care îl vom folosi:

```
Nume: Mike Artel
Adresa: 4264 Lovering Rd., Miami FL, 12777
Telefon: 075-961-0712
Email: mike_artel@venus.org
Interese: Drepturi civile,Educatie,Razboiul revolutionar
```

Pentru intrarea formatată prezentată mai sus, reprezentarea RTF este următoarea:

```
\b Nume: Mike Artel\b0\par
Adresa: 4264 Lovering Rd., Miami FL, 12777\par
Telefon: 075-961-0712\par
Email: mike_artel@venus.org\par
Interese: Drepturi civile,Educatie,Razboiul revolutionar\par
```

Pentru ca linia care conține numele să fie scrisă cu aldine, delimitați-o cu `\b` (urmată de un spațiu) pentru activarea scrierii cu aldine, respectiv `\b0` pentru a dezactiva această caracteristică. Fiecare linie conține un indicator de paragraf (`\par`) la sfârșit, pentru a indica procesorului de texte să se deplaseze pe linia următoare. Nu este foarte complicat.

Funcția de inițializare inserează limbajul de control RTF inițial (observați cele două backslash-uri pentru a obține un singur backslash în datele de ieșire):

```
sub rtf_init
{
    print "{\\rtf0\n";
    print "{\\fonttbl {\\f0 Times;}}\n;"
    print "\\plain \\f0 \\fs24\n";
}
```

Similar, funcția de curățenie inserează limbajul de control final (nu c-ar fi din cale-afară de mult!):

```
sub rtf_cleanup
{
    print "}\n";
}
```

Adevărata muncă este la formatarea intrării, dar chiar și această operație este relativ simplă. Principalele complicații apar la formatarea șirului de adrese și la determinarea liniilor de ieșire care trebuie afișate:

```
sub format_rtf_entry
{
    my ($entry_ref) = shift;
    my ($adresa);

        printf "\\b Nume: %s\\b0\\par\n", format_name ($entry_ref);
        $adresa = " ";
        $adresa .= $entry_ref->{strada} if $entry_ref->{strada};
        $adresa .= ", " . $entry_ref->{oras} if $entry_ref->{oras};
        $adresa .= ", " . $entry_ref->{stat} if $entry_ref->{stat};
        $adresa .= ", " . $entry_ref->{cod} if $entry_ref->{cod};
        print "Adresa: $adresa\\par\n" if $adresa;
        print "Telefon: $entry_ref->{telefon}\\par\n" if $entry_ref->{telefon}};
        print "Email: $entry_ref->{email}\\par\n" if $entry_ref->{email}};
        print "Interese: $entry_ref->{interese}\\par\n"
            if $entry_ref->{interese}};
        print "\\par\n";
}
```

Desigur, nu sunteți obligat să folosiți acest stil particular de formatare. Puteți schimba modul de afișare al oricăruia dintre câmpuri, astfel încât să fie posibilă modificarea stilului catalogului dumneavoastră tipărit aproape după dorință, prin simpla modificare a funcției `format_rtf_entry()`. Ținând cont de forma originală a catalogului (document creat cu procesorul de texte), nu este ceva chiar atât de simplu!

Scriptul `gen_cat` este acum complet. Putem genera catalogul în oricare din formatele de ieșire, prin rularea unor comenzi precum aceasta:

```
% gen_cat banquet > nume.txt
% gen_cat rtf > catalog.rtf
```

Sub Windows, pot rula `gen_cat`, iar fișierele sunt gata pentru a fi utilizate din interiorul unui procesor de texte bazat pe Windows. Sub UNIX, aș putea rula aceste comenzi și apoi îmi pot trimite mie personal fișierele de ieșire ca fișiere atașate la mesajele de e-mail, astfel încât să le pot prelua din calculatorul meu Macintosh și apoi să le încarc într-un procesor de texte. Se întâmplă să folosesc programul de poștă `mutt`, care permite specificarea fișierelor atașate în linia de comandă, folosind opțiunea `-a`. Îmi pot trimite mie însumi un mesaj cu ambele fișiere atașate, după cum urmează:

```
% mutt -a nume.txt -a catalog.rtf paul@snake.net
```

Și alte programe de poștă pot permite crearea fișierelor atașate. Alternativ, fișierele pot fi transferate și prin alte metode, cum ar fi FTP. În orice caz, după ce fișierele au ajuns acolo unde dorim, citirea listei numelor și inserția sa în documentul care conține programul pentru banchetul anual, respectiv citirea fișierului RTF în orice procesor de texte care înțelege formatul RTF, constituie operații simple.

DBI a facilitat extragerea informațiilor dorite din MySQL, iar funcționalitățile de prelucrare a textelor din Perl au simplificat inserția acelor informații în formatul pe care dorim să-l vedem. MySQL nu furnizează nici o modalitate specifică de formatare a datelor de ieșire, dar aceasta nu contează, datorită ușurinței cu care se pot integra caracteristicile sistemului MySQL de manipulare a bazelor de date într-un limbaj ca Perl, care dispune de excelente posibilități de manipulare a textelor.

Expedierea înștiințărilor pentru plata cotizațiilor

Când catalogul Ligii istorice este păstrat ca document creat cu un procesor de texte, determinarea membrilor care trebuie anunțați că trebuie să-și achite cotizațiile este o activitate consumatoare de timp și expusă la erori. Acum, când avem informațiile într-o bază de date, să vedem cum putem automatiza puțin procesul de înștiințare pentru plata cotizațiilor. Dorim să-i identificăm prin e-mail pe membrii care trebuie să-și achite cotizațiile, astfel încât să nu mai fie necesar să-i contactăm telefonic sau prin poșta clasică.

Ceea ce trebuie să facem este să determinăm membrii care sunt obligați să-și achite cotizația într-un anumit interval de timp. Interogarea pentru această operație necesită calculul unei date, ceea ce este relativ simplu:

```
SELECT ... FROM membru
WHERE data_expirare < DATE_ADD(CURRENT_DATE, INTERVAL limita DAY)
```

`limita` indică marja de zile pe care dorim să o acordăm pentru plata cotizației. Interogarea selectează intrările acelor membrilor care trebuie să-și achite cotizația într-un interval mai mic decât numărul de zile specificat. Ca un caz special, o valoare a limitei egală cu 0 descoperă membrii care trebuiau să-și achite cotizația în trecut (adică membrii... care nu mai sunt membri, de fapt).

După ce am identificat înregistrările care se pretează la notificări, ce facem cu ele? O opțiune ar fi expedierea unui mesaj direct din același script, dar este util să putem previzualiza lista înainte de a trimite mesajele. Din acest motiv, vom folosi o abordare în două etape:

- **Etapa 1:** Rulați un script denumit `need_renewal` (impune reînnoirea) pentru identificarea membrilor care trebuie să-și achite cotizația. Puteți pur și simplu examina această listă sau o puteți folosi ca date de intrare în a doua etapă, pentru a expedia înștiințările de plată a cotizației.
- **Etapa 2:** Rulați un script denumit `renewal_notify` (notificare pentru reînnoire), care trimite membrilor, prin e-mail, o înștiințare de plată a cotizației. Scriptul trebuie să indice membrii fără adrese de e-mail, astfel încât să îi puteți contacta prin alte mijloace.

Pentru prima parte a acestei sarcini, scriptul `need_renewal()` trebuie să-i identifice pe membrii care trebuie să-și achite cotizația. Acest lucru se poate face astfel:

```
# limita prestabilita este 30 de zile; reinitializeaza daca
# este dat un argument numeric
my ($limita) = 30;
$limita = shift(@ARGV) if @(ARGV) && ARGV[0] =~ /^\d+$/;

warn "Utilizeaza o limita de $limita zile\n";
my ($sth) = $dbh->prepare (qq{
   SELECT
      membru_id, email, nume, prenume, data_expirare,
      TO_DAYS (data_expirare) — TO_DAYS (CURRENT_DATE) AS zile
   FROM membru
   WHERE data_expirare < DATE_ADD(CURRENT_DATE, INTERVAL ? DAY)
   ORDER BY data_expirare, nume, prenume
});
$sth->execute ($limita);      # transfera limita ca valoare
                              # pentru campul de inlocuire

while (my $hash_ref = $sth->fetchrow_hashref ())
{
   print join ("\t",
            $hash_ref->{membru_id},
            $hash_ref->{email},
            $hash_ref->{nume},
            $hash_ref->{prenume},
            $hash_ref->{data_expirare},
            $hash_ref->{zile} . " zile"
      . "\n";
}
$sth->finish ();
```


Datele de ieșire ale scriptului `need_renewal` se prezintă oarecum astfel (dumneavoastră veți obține rezultate diferite, deoarece rezultatele sunt determinate în raport cu data curentă, care va fi diferită pentru dumneavoastră, când citiți această carte, în raport cu data la care a fost scrisă cartea):

```
89   g.steve@pluto.com       Garner    Steve   1999-08-03   -32 zile
18   york_mark@earth.com     York      Mark    1999-08-24   -11 zile
82   john_edwards@venus.org  Edwards   John    1999-09-12   8 zile
```

Observați că unii membri trebuie să-și achite cotizația într-un număr negativ de zile, ceea ce înseamnă că nu mai sunt membri! (Asta se întâmplă când folosiți o evidență manuală a înregistrărilor; oamenii scapă printre degete. Acum, când avem informațiile într-o bază de date, descoperim că am omis câteva persoane înainte.)

Cea de-a doua parte a sarcinii de înștiințare privind plata cotizațiilor implică utilizarea unui script `renewal_notify`, care expediază notificări prin e-mail. Pentru a facilita puțin utilizarea acestui script, îl putem determina să „înțeleagă" trei categorii de argumente din linia de comandă: numere de identificare a membrilor, adrese de e-mail și nume de fișiere. Argumentele numerice indică valorile identificatorilor de membru, iar argumentele care conțin caracterul `@` semnifică adrese de e-mail. Orice alte argumente sunt interpretate ca fiind numele unui fișier care trebuie citit pentru a se găsi numerele de identificare sau adresele de e-mail. Această metodă vă permite să specificați membrii în funcție de numărul de identificare sau de adresa de e-mail, ceea ce puteți face fie direct din linia de comandă, fie prin menționarea lor într-un fișier. (În particular, puteți folosi datele de ieșire ale scriptului `need_renewal` ca date de intrare pentru scriptul `renewal_notify`.)

Pentru fiecare membru care urmează a primi o înștiințare, scriptul caută intrarea corespunzătoare din tabelul `membru`, extrage adresa de e-mail și trimite un mesaj la adresa respectivă. Dacă nu există nici o adresă de e-mail în intrarea respectivă, scriptul `renewal_notify` generează un mesaj, pentru a vă avertiza că acești membri trebuie contactați într-un alt mod.

Pentru a trimite mesaje de e-mail, scriptul `renewal_notify` deschide un canal spre programul `sendmail` și determină trecerea mesajului de e-mail prin canalul respectiv.[2] Calea de acces spre programul `sendmail` este configurată ca parametru aproape de începutul scriptului. Poate că veți fi obligat să modificați această cale, deoarece amplasarea programului `sendmail` variază de la un sistem la altul:

```
# schimba calea conform configuratiei sistemului dumneavoastra
my ($sendmail) = "/usr/lib/sendmail -t -oi";
```

Ciclul principal de prelucrare a argumentului funcționează după cum urmează. Dacă în linia de comandă nu au fost specificate argumente, citim ca date de intrare datele de intrare standard (STDIN). În caz contrar, prelucrăm fiecare argument transferându-l funcției `interpret_argument()`, pentru a fi clasificat ca număr de identificare, adresă de e-mail sau nume de fișier:

```
if (@ARGV = = 0)    # nu sunt argumente, citeste STDIN
                    # pentru a obtine valorile
```

Continuare

[2] Acest procedeu nu este funcțional pentru Windows, care nu dispune de programul `sendmail`. Va trebui să găsiți un modul de expediere a mesajelor de e-mail și să-l folosiți pe acela. – N.A.

Continuare

```
{
   read_file (\*STDIN);
}
else
{
   while (my @arg = shift (@ARGV))
   {
      # interpreteaza argumentul, cu recursivitatea numelui de fisier
      interpret_argument ($arg, 1);
   }
}
```

Funcția `read_file()` citește conținutul unui fișier (care se presupune că este deja deschis) și examinează primul câmp al fiecărei linii. (Dacă alimentăm scriptul `renewal_notify` cu datele de ieșire ale scriptului `need_renewal`, fiecare linie are mai multe câmpuri, dar noi dorim să o examinăm numai pe prima.)

```
sub read_file
{
my ($fh) = script;
my ($arg);

   while (defined ($arg = <$fh>))
   {
      # elimina tot ce este plasat dincolo de coloana 1,
      # inclusiv caracterul linie noua
      $arg =~ s/\s.*//s;
      # interpreteaza argumentul, fara recursivitatea
      # numelui de fisier
      interpret_argument ($arg, 0);
   }
}
```

Funcția `interpret_argument()` clasifică fiecare argument pentru a determina dacă este un număr de identificare, o adresă de e-mail sau un nume de fișier. Pentru numerele de identificare și adresele de e-mail, caută în intrarea de membru corespunzătoare și o transferă scriptului `notify_member()`. Trebuie să fim atenți la membrii specificați în funcție de adresa de e-mail. Este posibil ca doi membri să aibă aceeași adresă (de exemplu soțul și soția) și nu dorim să trimitem un mesaj cuiva pentru care mesajul respectiv nu este valabil. Pentru a evita această situație, căutăm identificatorul de membru care corespunde unei adrese de e-mail, pentru a ne asigura că este unul singur. Dacă adresa corespunde mai multor numere de identificare, avem un echivoc și îl vom ignora, după afișarea unui mesaj de avertizare.

Dacă un argument nu apare sub forma unui număr de identificare sau al unei adrese de e-mail, se presupune că este numele unui fișier care trebuie citit pentru obținerea altor date de intrare. Și în acest caz trebuie să fim atenți – nu dorim să citim un fișier dacă deja citim un alt fișier, pentru a evita posibilitatea unui ciclu infinit:


```
sub interpret_argument
{
my ($arg, $recurse) = @_;
my ($query, $ary_ref);

   if ($arg =~ /^\d+$/)          # identificatorul numeric de membru
   {
      notify_member ($arg);
   }
   elsif ($arg=~ /@/)            # adresa de e-mail
   {
      # obtine identificatorul de membru asociat cu adresa
      # (trebuie sa fie unul singur)
      $query =qq{ SELECT membru_id FROM membru WHERE email = ? };
      $ary_ref = $dbh->selectcol_arrayref ($query, undef, $arg);
      if (scalar (@{ary_ref}) == 0)
      {
           warn "Adresa de e-mail $arg nu corespunde nici unei
           intrari: se ignora\n";
      }
      elsif (scalar (@{ary_ref}) > 1)
      {
           warn "Adresa de e-mail $arg corespunde mai multor
           intrari: se ignora\n";
      }
      else
      {
           notify_member ($ary_ref->[0]);
      }
   }
   else                          # nume de fisier
   {
      if (!$recurse)
      {
           warn "numele de fisier $arg se afla in interiorul
           unui fisier: se ignora\n";
      }
      else
      {
           open (IN, $arg) or die "Nu poate deschide $arg: $!\n";
```

Continuare

Continuare

```
            read_file (\*IN);
            close (IN);
        }
    }
}
```

Programul aferent funcției `notify_member()`, care trimite efectiv înștiințarea de plată a cotizației, este prezentat în continuare. Dacă se dovedește că persoana membru nu are adresă de e-mail, nu se poate face nimic, dar `notify_member()` afișează un avertisment, astfel încât să știți că trebuie să contactați persoana respectivă într-un alt mod. Puteți invoca `show_member` cu numărul de identificare a persoanei membru prezentat în mesaj, pentru a vedea intrarea în totalitate – pentru a afla, de exemplu, care este numărul de telefon și adresa persoanei membru.

```
# anunta membrul ca trebuie sa-si plateasca in curand cotizatia

sub notify_member
{
my ($membru_id) = shift;
my ($query, $sth, $entry_ref, @col_name);

   warn "Anunta $membru_id...\n";
   $query = qq{ SELECT * FROM membru WHERE membru_id = ? };
   $sth = $dbh->prepare ($query);
   $sth->execute ($membru_id);
   @col_name = @{$sth->{NAME}};
   $entry_ref = $sth->fetchrow_hashref ();
   $sth->finish ();
   if (!$entry_ref)          # nu a fost gasit nici un membru!
   {
      warn " Nu a fost gasita
      NICI O INTRARE pentru membrul $member_id!\n";
      return;
   }
   open (OUT, " | $sendmail")
      or die "Nu poate deschide programul de posta\n";
   print OUT <<EOF;
To: $entry_ref->{email}
Subject: Necesitatea platii cotizatiei la Liga istorica
Buna ziua. Va trebui ca in curand sa va achitati cotizatia de membru al
Ligii istorice. Speram ca veti gasi cateva minute pentru a contacta
secretariatul Ligii în vederea achitarii cotizatiei. Continutul intrarii
dumneavoastra din catalogul cu membri este prezentat mai jos. Va rugam
sa retineti mai ales data expirarii.
Multumim.
```



```
EOF
   foreach my $col_name (@col_name)
   {
      printf OUT "%s: %s\n", $col_name, $entry_ref->{$col_name};
   }
   close (OUT);
}
```

Puteți realiza chiar și mesaje mai atrăgătoare; de exemplu, puteți adăuga o coloană la tabelul `membru` care să conțină data trimiterii ultimei înștiințări. Astfel, nu veți trimite înștiințări prea frecvent. În forma în care este acum, vom presupune că nu veți rula programul mai frecvent decât o dată pe lună.

Cele două scripturi sunt acum terminate, deci le puteți folosi astfel:

```
% need_renewal > junk
% (examinati fisierul junk pentru a vedea daca arata bine)
% renewal_notify junk
```

Pentru a trimite notificări membrilor individuali, îi puteți specifica în funcție de numărul de identificare sau de adresa de e-mail:

```
% need_renewal 18 g.steve@pluto.com
```

Editarea intrărilor corespunzătoare membrilor Ligii istorice

După ce începem să trimitem înștiințări de plată a cotizației, putem presupune că unele din persoanele pe care le-am anunțat își vor achita cotizația. Când se va întâmpla acest lucru, vom avea nevoie de o modalitate de a actualiza intrările lor cu date noi de expirare. În capitolul următor, vom crea o modalitate de a edita înregistrările membrilor dintr-un browser Web, dar aici vom crea un script în linie de comandă `edit_member`, care vă permite să actualizați intrările folosind o metodă simplă, de a solicita noile valori pentru fiecare componentă a intrării. Scriptul funcționează astfel:

- Dacă este invocat fără argumente în linia de comandă, `edit_member` presupune că doriți să introduceți un membru nou, solicită informațiile inițiale care vor fi amplasate în intrarea corespunzătoare membrului și creează o intrare nouă.
- Dacă este invocat cu un identificator numeric de membru în linia de comandă, `edit_member` caută conținutul curent al intrării, apoi solicită actualizări pentru fiecare coloană în parte. Dacă introduceți o valoare pentru o coloană, înlocuiește valoarea curentă. Dacă apăsați pe tasta Enter, conținutul coloanei nu se modifică. (Dacă nu cunoașteți numărul de identificare al unui membru, puteți rula `show_member nume`, pentru a afla care este acesta.)

Probabil că editarea unei întregi intrări în acest mod este o muncă inutilă, dacă nu doriți decât să actualizați data plății cotizației unui membru. Pe de altă parte, un script ca acesta furnizează și o modalitate simplă, universală, de a actualiza orice componentă a unei intrări fără a cunoaște limbajul SQL. (Un caz special este acela în care `edit_member` nu vă va permite să modificați câmpul `membru_id`, deoarece acesta este atribuit în mod automat la crearea unei intrări și nu trebuie modificat ulterior.)

Prima informație pe care `edit_member` trebuie să o cunoască rezidă în numele coloanelor din tabelul `membru`:

```
# obtine numele coloanelor din tabelul membru
my ($sth) = $dbh->prepare (qq{ SELECT * FROM membru WHERE 1 = 0 });
$sth->execute ();
my (@col_name) = @{$sth->{NAME}};
$sth->finish ();
```

Apoi, putem introduce ciclul principal:

```
if (@ARGV == 0)     # daca nu exista argumente,
                    # creeaza o intrare noua
{
   new_member (\@col_name);
}
else                # in caz contrar, editeaza intrarile folosind
                    # argumente precum identificatorii de membru
{
my (@id);

   # salveaza @ARGV, apoi goleste-l pentru a citi din STDIN
   # nu foloseste argumente precum numele de fisiere
   @id = @ARGV;
   @ARGV = ();
   # pentru fiecare valoare a identificatorului,
   # cauta intrarea si apoi o editeaza
   while (my $id = shift (@id))
   {
      my ($entry_ref);

         $sth = $dbh->prepare (qq{
                    SELECT * FROM membru WHERE membru_id = ?
                 });
         $sth->execute ($id);
         $entry_ref = $sth->fetchrow_hashref ();
         $sth->finish ();
         if (!$entry_ref)
         {
                warn "Nici un membru nu are identificatorul = $id\n";
                next;
         }
         edit_member (\@col_name, $entry_ref);
   }
}
```


Programul pentru crearea unei noi intrări de membru este prezentat în continuare. Programul solicită valori pentru fiecare coloană a tabelului `membru`, apoi emite o instrucțiune `SELECT` pentru a adăuga o înregistrare nouă:

```
# creeaza o noua inregistrare de membru

sub new_member
{
my ($col_name_ref) = shift;
my ($entry_ref);
my ($col_val, $query, $delim);

   return unless prompt ("Creez o intrare noua? ") =~ /^y/i;
   # cere valori noi; utilizatorul tasteaza valori noi, "null" pentru a
   # introduce o valoare NULL, "exit" pentru a parasi programul prematur.
   foreach my ($col_name) (@{$col_name_ref})
   {
      next if $col_name eq "membru_id"; # ignora campul cheie
      $col_val = col_prompt ($col_name, "", 0);
      next if $col_val eq "";          # utilizatorul a apasat Enter
      return if $col_val =~ /^exit$/i; # inchidere prematura
      $col_val = undef if $col_val =~ /^null$/i;
      $entry_ref->{$col_name} = $col_val;
   }
   # afiseaza valorile, cere confirmarea inainte de insertie
   show_member ($col_name_ref, $entry_ref);
   return unless prompt ("\nInserez aceasta intrare? ") =~ /^y/i;

   # construieste o interogare INSERT, apoi o emite.
   $query = "INSERT INTO membru";
   $delim = " SET ";          # pune "SET" inaintea primei coloane, ","
                              # inaintea celorlalte
   foreach my $col_name (@{col_name_ref})
   {
      # specifica valori numai pentru coloane care au primit o valoare
      next if !defined ($entry_ref->{$col_name});
      # functia quote() citeaza undef drept cuvantul NULL,
      # (fara ghilimele), adica exact ceea ce dorim.
      $ query .= sprintf ("%s %s=%s", $delim, $col_name,
                          $dbh->quote ($entry_ref->{$col_name}));
      $delim = ",";
   }
   warn "Avertisment: intrarea nu a fost inserata?\n"
      unless $dbh->do ($query) == 1;
}
```

Rutinele de solicitare utilizate de `new_member()` se prezintă astfel:

```
# pune o intrebare, cere un raspuns

sub prompt
{
my ($str) = shift;

   print STDERR $str;
   chomp ($str = <STDIN>);
   return ($str);
}

# solicita valoarea unei coloane; afiseaza valoarea curenta
# in solicitare daca $show_current are valoarea "adevarat"

sub col_prompt
{
my ($name, $val, $show_current) = @_;
my ($prompt, $str);

   $prompt = $name;
   $prompt .= " [$val]" if $show_current;
   $prompt .= ": ";
   print STDERR $prompt;
   chomp ($str = <STDIN>);
   return ($str ? $str : $val);
}
```

Motivul pentru care funcția `col_prompt()` preia un argument `$show_current` este acela că vom folosi această funcție și pentru solicitarea valorilor coloanelor din intrările existente aferente membrilor atunci când scriptul este utilizat pentru actualizarea unei intrări. `$show_current` va fi 0 când creăm o intrare nouă, deoarece nu există valori curente de afișat. În acest din urmă caz, promptul va afișa valoarea curentă, pe care utilizatorul o poate accepta, pur și simplu, prin apăsarea tastei Enter.

Programul pentru editarea intrării aferente unui membru existent este similară cu aceea pentru crearea unui membru nou. Totuși, avem o intrare de lucru, deci rutina de solicitare afișează valorile intrării curente, iar funcția `edit_member()` emite o instrucțiune `UPDATE`, nu o instrucțiune `INSERT`:

```
# editeaza continutul curent al unei intrari

sub edit_member
{
my ($col_name_ref, $entry_ref) =@_;
my ($col_val, $query, $delim);
```



```
   # afiseaza valorile initiale, cere permisiunea de a continua
   # si de a edita
   show_member ($col_name_ref, $entry_ref);
   return unless prompt ("\nEditez aceasta intrare? ") =~ /^y/i;
   # solicita valori noi; utilizatorul tasteaza valoarea noua
   # pentru a inlocui valoarea existenta, "null" pentru a
   # introduce o valoare NULL, "exit" pentru a parasi prematur
   # programul, respectiv Enter pentru a accepta valoarea existenta.
   foreach my $col_name (@{$col_name_ref})
   {
      next if $col_name eq "membru_id"; # ignora campul cheie
      $col_val = $entry_ref->{$col_name};
      $col_val = "NULL" unless defined ($col_val);
      $col_val = col_prompt ($col_name, $col_val, 1);
      return if $col_val =~ /^exit$/i; # inchidere prematura
      $col_val = undef if $col_val =~ /^null$/i;
      $entry_ref->{$col_name} = $col_val;
   }
   # afiseaza valorile noi, cere confirmare inainte de actualizare
   show_member ($col_name_ref, $entry_ref);
   return unless prompt ("\nActualizez aceasta intrare? ") =~ /^y/i;

   # construieste o interogare UPDATE, iar apoi o emite.
   $query = "UPDATE membru";
   $delim = " SET ";          # pune "SET" inaintea primei coloane, ","
                              # inaintea celorlalte
   foreach my $col_name (@{col_name_ref})
   {
      next if $col_name eq "membru_id";       # ignora campul cheie
      # functia quote() citeaza undef drept cuvantul NULL
      # (fara ghilimele), adica exact ceea ce dorim.
      $ query .= sprintf ("%s %s=%s", $delim, $col_name,
                        $dbh->quote ($entry_ref->{$col_name}));
      $delim = ",";
   }
   $query = " WHERE membru_id = ?";
   warn "Avertisment: intrarea nu a fost actualizată?\n"
      unless $dbh->do ($query, undef, $entry_ref->{membru_id}) == 1;
}
```

O problemă a funcției edit_member este aceea că nu execută validări ale valorilor de intrare. Pentru majoritatea câmpurilor din tabelul membru, nu prea aveți ce valida – acestea nu sunt nimic altceva decât câmpuri șir. Dar trebuie verificate valorile datelor de intrare pentru coloana data_expirare, pentru a exista siguranța că sunt date și nu altceva. Într-o aplicație de uz general pentru introducerea datelor, probabil că veți dori să extrageți informații despre un tabel, pentru a determina tipurile tuturor coloanelor sale. Apoi, puteți să formulați restricții de validare pe baza acestor tipuri, ceea ce este prea complicat pentru ceea ce vreau eu să prezint aici, deci voi adăuga un artificiu rapid la funcția col_prompt() pentru a verifica formatul datelor de intrare, în cazul în care numele coloanei este "data_expirare". O verificare minimală a valorii datei se poate face astfel:

```
sub col_prompt
{
my ($name, $val, $show_current) = @_;
my ($prompt, $str);

loop:
   $prompt = $name;
   $prompt .= " [$val]" if $show_current;
   $prompt .= ": ";
   print STDERR $prompt;
   chomp ($str = <STDIN>);
   # executa o verificare rudimentara a datei de expirare
   if ($str && $nume eq "data_expirare")       # verifica formatul
                                               # datei de expirare
   {
      $str =~ /^\d+[^\d]\d+[^\d]\d+$/
           or goto loop;
   }
   return ($str ? $str : $val);
}
```

Modelul testează cele trei secvențe de cifre separate prin caractere non-cifrice. Aceasta este numai o verificare parțială, deoarece nu detectează valori precum "1999-14-92" ca fiind incorecte. Pentru a îmbunătăți scriptul, puteți insera verificări mai riguroase ale datelor și alte verificări, precum obligativitatea existenței unor valori nevide în câmpurile care conțin numele și prenumele.

Alte îmbunătățiri pot consta din omiterea actualizării dacă nu există coloane modificate, respectiv de a anunța utilizatorul dacă înregistrarea a mai fost modificată de altcineva în timp ce utilizatorul respectiv o edita. Puteți face aceasta salvând valorile originale ale coloanelor care compun intrările membrilor și apoi scriind instrucțiunea UPDATE pentru a actualiza numai acele coloane care s-au modificat. Dacă nu s-a modificat nici o coloană, instrucțiunea nici măcar nu mai trebuie emisă. De asemenea, se poate scrie clauza WHERE astfel încât să includă AND nume_coloana = valoare_coloana pentru fiecare valoare originală din coloană. Acest procedeu va cauza eșecul instrucțiunii UPDATE dacă

o altă persoană a modificat înregistrarea, ceea ce reprezintă un indiciu al faptului că două persoane încearcă să modifice intrarea simultan.

Identificarea membrilor cu interese comune ai Ligii istorice

Una din îndatoririle secretarului Ligii istorice este de a prelucra cererile provenite de la membrii care doresc o listă cu alți membri care manifestă un interes particular într-un anumit episod din istoria Statelor Unite, cum ar fi Marea Criză sau viața lui Abraham Lincoln. Localizarea unor asemenea membri este destul de ușoară când catalogul se află sub forma unui document creat cu un procesor de texte, folosind funcția Find (găsire) a procesorului de texte. Totuși, generarea unei liste care să fie compusă numai din intrările corespunzătoare ale membrilor este mai dificilă, deoarece implică o mulțime de operații de copiere și lipire. Cu MySQL, această operație devine cu mult mai simplă, deoarece putem rula o interogare astfel:

```
SELECT * FROM member WHERE interese LIKE "%lincoln%"
ORDER BY nume, prenume
```

Din păcate, rezultatele nu arată prea frumos dacă rulăm această interogare din clientul mysql. Să alcătuim un mic script DBI, denumit interese. Mai întâi, scriptul verifică pentru a se asigura că există cel puțin un argument denumit în linia de comandă, deoarece în caz contrar nu dispune de criterii de căutare. Apoi, pentru fiecare argument, scriptul rulează o căutare având drept criteriu coloana interese a tabelului membru:

```
@ARGV or die: "Utilizare: cuvantul cheie interese\n";
cauta_membri (shift (@ARGV)) while @ARGV;
```

Pentru a căuta șirul determinat de cuvântul cheie, inserăm caractere de înlocuire % (procent) la fiecare capăt al acestuia, astfel încât șirul să poată fi găsit oriunde în coloana interese. Apoi, afișăm intrările care satisfac criteriul de căutare:

```
sub cauta_membri
{
my ($interese) = shift;
my ($sth, $numar);

   print "Cauta rezultatele pentru cuvantul cheie: $interes\n\n\";
   $sth = $dbh->prepare (qq{
           SELECT * FROM membru WHERE interese LIKE ?
           ORDER BY nume, prenume
     });
   # cauta sirul oriunde in campul "interese"
   $sth->execute ("%" . $interese . "%");
   $numar = 0;
   while (my $hash_ref = $sth->fetchrow_hashref ())
   {
      format_entry ($hash_ref);
      ++$numar;
   }
   print "Au fost gasite $numar intrari\n\n";
}
```

Funcția `format_entry()` nu este prezentată aici. În esență, este asemănătoare cu funcția `format_rtf_entry()` din scriptul gen_cat, dar fără cuvintele de control RTF.

Publicarea pe Internet a catalogului Ligii istorice

În secțiunea următoare, „Utilizarea DBI în aplicațiile Web", vom începe să scriem scripturi care se conectează la serverul MySQL pentru a extrage informații și pentru a scrie acele informații în forma paginilor Web care apar în browserul Web al unui client. Aceste scripturi generează cod HTML în mod dinamic, în conformitate cu cerințele clientului. Înainte de a ajunge la acel punct, să începem să ne gândim la HTML inserând un program DBI care generează un document HTML static, ce poate fi încărcat în arborele cu documente al unui server Web. Un candidat bun în acest sens este un catalog al Ligii istorice pe care îl creați în format HTML, deoarece unul din scopurile noastre era oricum publicarea pe Internet a catalogului.

În general, un document HTML are o structură care este oarecum asemănătoare cu următoarea:

```
<HTML>                              ⇦ începutul documentului
<HEAD>                              ⇦ începutul capului documentului
<TITLE>Titlul paginii mele</TITLE>  ⇦ titlul documentului
</HEAD>                             ⇦ sfârșitul capului documentului
<BODY>                              ⇦ începutul corpului documentului
<H1>Titlu de nivel 1</H1>           ⇦ un titlu de nivel 1

... continutul corpului documentului ...

</BODY>                             ⇦ sfârșitul corpului documentului
</HTML>                             ⇦ sfârșitul documentului
```

Pentru a genera catalogul în acest format, nu este necesar să scrieți un întreg script. Vă reamintiți că, atunci când am scris scriptul gen_cat, am folosit un cadru extensibil astfel încât să putem insera programul pentru generarea catalogului în formate suplimentare. Aceasta înseamnă că, pentru a adăuga programe pentru generarea de date de ieșire în format HTML, trebuie să scriem funcțiile de inițializare și „curățenie" a documentului, precum și o funcție pentru formatarea intrărilor individuale. Apoi, trebuie să creăm un element de tip „cutie de distribuție" care să indice spre aceste funcții.

Schița de document prezentată anterior se poate împărți destul de simplu în secțiuni de prolog și epilog, care pot fi manipulate de funcțiile de inițializare, respectiv curățenie, precum și într-o parte de mijloc, care poate fi generată de funcția de formatare a intrărilor. Funcția de inițializare HTML generează toate elementele documentului, până la titlul de nivel 1, iar funcția de curățenie generează etichetele de închidere </BODY> și </HTML>:

```
sub html_init
{
   print "<HTML>\n";
```



```
   print "<HEAD>\n";
   print "<TITLE>Catalogul membrilor Ligii istorice</TITLE>\n";
   print "</HEAD>\n";
   print "<BODY>\n";
   print "<H1>Catalogul membrilor Ligii istorice</H1>\n";
}

sub html_cleanup
{
   print "</BODY>\n";
   print "</HTML>\n";
}
```

Adevărata muncă, de obicei, rezidă în formatarea intrărilor. Dar nici măcar aceasta nu este prea grea. Putem copia funcția `format_rtf_entry()`, ne asigurăm că toate caracterele speciale din cadrul intrării sunt codificate și înlocuim cuvintele de control RTF cu etichetele de marcare HTML:

```
sub format_html_entry
{
my ($entry_ref) = shift;
my ($adresa);

   # codeaza caracterele speciale din HTML
   foreach my $key (keys (%{entry_ref}))
   {
      $entry_ref->{key} =~ s/&/&amp;/g;
      $entry_ref->{key} =~ s/\"/&quot;/g;
      $entry_ref->{key} =~ s/>/&gt;/g;
      $entry_ref->{key} =~ s//&lt;/g;
   }
   printf "<STRONG> Nume: %s</STRONG><BR>\n",
          format_name ($entry_ref);
   $adresa = " ";
   $adresa .= $entry_ref->{strada} if $entry_ref->{strada};
   $adresa .= ", " . $entry_ref->{oras} if $entry_ref->{oras};
   $adresa .= ", " . $entry_ref->{stat} if $entry_ref->{stat};
   $adresa .= " " . $entry_ref->{cod} if $entry_ref->{cod};
   print "Adresa: $adresa<BR>\n" if $adresa;
   print "Telefon: $entry_ref->{telefon}<BR>\n"
         if $entry_ref->{telefon};
   print "Email: $entry_ref->{email}<BR>\n"
         if $entry_ref->{email};
   print "Interese: $entry_ref->{interese}<BR>\n"
         if $entry_ref->{interese};
   print "</BR>n";
}
```

Acum, vom adăuga la cutia de distribuție un alt element, care indică spre funcțiile de scriere HTML, iar modificările aduse scriptului gen_cat au fost finalizate:

```
# cutie de distributie care contine functii de formatare
# pentru fiecare format al datelor de iesire
my (%cutie_distributie) =
  (
    "banquet"       =>          # functii pentru lista de banchet
    {
        "init"      => undef,   # nu este necesara nici
                                # o initializare
        "entry"     => \&format_banquet_entry,
        "cleanup"   => undef    # curatenia nu este necesara
    },
    "rtf" =>                    # functii pentru formatul RTF
    {
        "init"      => \&rtf_init,
        "entry"     => \&format_rtf_entry,
        "cleanup"   => \&rtf_cleanup
    },
    "html" =>                   # functii pentru formatul HTML
    {
        "init"      => \&html_init,
        "entry"     => \&format_html_entry,
        "cleanup"   => \&html_cleanup
    }
  );
```

Pentru a genera catalogul în format HTML, rulați următoarea comandă și instalați fișierul de ieșire rezultant în arborele cu documente al serverului dumneavoastră Web:

```
% gen_cat html > catalog.html
```

Când actualizați catalogul, puteți rula comanda din nou, pentru a actualiza versiunea electronică a acestuia. O altă strategie este să configurați o sarcină a utilitarului cron care să se execute periodic. Astfel, catalogul în versiune electronică va fi actualizat automat. De exemplu, pot folosi o intrare crontab ca aceasta, pentru a rula gen_cat în fiecare dimineață la ora 4:

```
0 4 * * * /u/paul/samp_db/gen_cat > /usr/local/apache/htdocs/catalog.html
```

Utilizatorul în numele căruia rulează această sarcină cron trebuie să aibă atât permisiunea de a executa scripturile localizate în catalogul meu samp_db, cât și permisiunea de a scrie fișiere în arborele cu documente al serverului Web.

Utilizarea DBI în aplicații Web

Scripturile DBI pe care le-am scris până acum au fost concepute pentru a fi utilizate din interpretor într-un mediu bazat pe linii de comandă, dar DBI este util și în alte contexte, cum ar fi în cazul dezvoltării aplicațiilor bazate pe Web. Când scrieți scripturi DBI care pot fi invocate din browserul dumneavoastră Web, deschideți noi și interesante posibilități de a interacționa cu bazele dumneavoastră de date.

De exemplu, dacă afișați date în format tabelar, puteți transforma cu ușurință fiecare titlu de coloană într-o legătură (link), pe care o puteți selecta pentru a sorta din nou datele din coloana respectivă. Aceasta vă permite să vizualizați datele într-un alt mod cu un singur clic, fără a introduce nici o interogare. Sau, puteți furniza un formular în care un utilizator poate introduce criterii pentru căutarea într-o bază de date, iar apoi afișați o pagină care conține rezultatele căutării. Funcționalități simple ca aceasta pot avea un impact semnificativ asupra nivelului de interactivitate pe care îl furnizați pentru accesul la conținutul bazelor dumneavoastră de date. În plus, de regulă caracteristicile de afișare ale browserelor Web sunt superioare celor ale unei ferestre terminal, deci datele de ieșire arată și ele frecvent mai bine.

În această secțiune, vom crea următoarele scripturi bazate pe Web:

- **Un browser general pentru tabelele din baza de date** samp_db. Această aplicație nu este legată de vreo operație specifică pe care dorim să o efectuăm cu baza de date, dar ilustrează numeroase concepte de programare în Web și furnizează o modalitate convenabilă de examinare a informațiilor pe care le conțin tabelele.
- **Un browser de punctaje, care ne permite să examinăm rezultatele obținute la orice test sau chestionar.** Este un mijloc rapid de examinare a rezultatelor de la evenimentele de tip examinare și este util atunci când trebuie să stabilim curba rezultatelor la un test, astfel încât să putem acorda lucrărilor note de tip literal.
- **Un script pentru identificarea membrilor Ligii istorice care manifestă interese comune.** Această operație se poate realiza permițând utilizatorului să introducă o expresie de căutare, iar apoi căutând expresia respectivă în câmpul interese al tabelului membru. Deja am scris un script în linie de comandă care execută această operație, dar o versiune bazată pe Web furnizează un punct de referință instructiv, permițând o comparație între două abordări ale aceleiași sarcini.

Vom scrie aceste scripturi folosind modulul Perl CGI.pm, care reprezintă modalitatea cea mai simplă de a lega scripturi DBI la Web. (Pentru instrucțiuni cu privire la procurarea modulului CGI.pm, vezi Anexa A.) Modulul CGI.pm este astfel denumit deoarece vă ajută să scrieți scripturi care folosesc protocolul Common Gateway Interface, care definește modul în care un server Web comunică cu alte programe. Modulul CGI.pm se ocupă de detaliile pe care le incumbă un număr de operații comune de întreținere, cum ar fi colectarea valorilor parametrilor transferați scriptului dumneavoastră de către serverul Web ca date de intrare. De asemenea, CGI.pm furnizează metode convenabile de generare a datelor de ieșire în format HTML, ceea ce reduce posibilitatea de a scrie programe HTML cu malformații în comparație cu a scrie personal etichete HTML brute.

Veţi învăţa suficient de multe despre CGI.pm în capitolul de faţă pentru a vă scrie propriile dumneavoastră aplicaţii Web, dar, desigur, nu am descris toate funcţionalităţile sale. Pentru a învăţa mai multe despre acest modul, consultaţi volumul *Official Guide to Programming with CGI.pm*, de Lincoln Stein (John Wiley, 1998) sau examinaţi documentaţia electronică la adresa:

```
http://stein.cshl.org/WWW/software/CGI/
```

Configurarea serverului Apache pentru scripturile CGI

În afară de DBI şi CGI.pm, mai avem nevoie de încă o componentă pentru scrierea scripturilor bazate pe Web, şi anume un server Web. Instrucţiunile din această secţiune sunt orientate în direcţia utilizării scripturilor cu serverul Apache, dar probabil că puteţi folosi un alt server, dacă doriţi, printr-o oarecare adaptare a instrucţiunilor.

Diferitele componente ale unei instalări a serverului Apache se găsesc, de obicei, în catalogul `/usr/local/apache`. Pentru expunerea de faţă, cele mai importante subcataloage ale acestui catalog sunt `htdocs` (pentru arborele de documente HTML), `cgi-bin` (pentru scripturile executabile şi programele care vor fi invocate de către serverul Web) şi `conf` (pentru fişierele de configurare). Aceste cataloage pot avea un alt amplasament pe sistemul dumneavoastră. În acest caz, operaţi modificările adecvate la notele care urmează.

Trebuie să vă asiguraţi că `cgi-bin` nu se află în interiorul arborelui de documente Apache, astfel încât scripturile din cadrul său să nu poată fi solicitate sub formă de text simplu. Aceasta este o măsură de precauţie. Nu doriţi ca programe client rău intenţionate să vă examineze scripturile în căutarea unor breşe de securitate, prin „aspirarea" textului scripturilor şi studierea acestuia.

Pentru a instala un script CGI în vederea utilizării cu serverul Apache, inseraţi-l în catalogul `cgi-bin`, după care transferaţi dreptul de proprietate a fişierului către utilizatorul pentru care rulează serverul Apache şi modificaţi-i modul astfel încât să fie executabil şi să poată fi citit numai de către acel utilizator. De exemplu, dacă serverul Apache rulează ca un utilizator cu numele www, folosiţi următoarele comenzi:

```
% chown www nume_script
% chmod 500 nume_script
```

Probabil că va trebui să rulaţi aceste comenzi ca www sau ca `root`. Dacă nu aveţi permisiunea de a instala scripturi în catalogul `cgi-bin`, puteţi cere administratorului de sistem să efectueze această operaţie în numele dumneavoastră.

După ce scriptul a fost instalat, îl puteţi solicita din browserul dumneavoastră expediind serverului Web adresa URL adecvată. În mod caracteristic, adresa URL se prezintă astfel:

```
http://numele.gazdei.dv/cgi-bin/nume_script
```

Solicitarea scriptului din browserul dumneavoastră de Web determină execuţia acestuia de către serverul Web. Datele de ieşire ale scriptului vă sunt trimise înapoi, iar rezultatul apare ca o pagină Web în browserul dumneavoastră.

Dacă doriţi să folosiţi scripturi CGI cu `mod_perl` pentru a obţine performanţe mai bune, iată ce aveţi de făcut:

1. Verificaţi dacă dispuneţi cel puţin de următoarele versiuni ale programelor necesare: Perl 5.004, CGI.pm 2.36 şi `mod_perl` 1.07.
2. Asiguraţi-vă că `mod_perl` este compilat în executabilul dumneavoastră Apache.
3. Configuraţi un catalog pentru stocarea scripturilor. Eu folosesc `/usr/local/apache/cgi-perl`. Catalogul `cgi-perl` nu trebuie să se afle în interiorul arborelui de documente al serverului dumneavoastră Apache, din aceleaşi motive de securitate ca şi catalogul `cgi-bin`.
4. Cereţi serverului Apache să asocieze scripturile care se găsesc în catalogul `cgi-perl` cu `mod_perl`:

```
Alias /cgi-perl/ /usr/local/apache/cgi-perl

<Location /cgi-perl>
   SetHandler perl-script
   PerlHandler Apache::Registry
   Options ExecCGI
</Location>
```

Dacă folosiţi o versiune curentă a serverului Apache care utilizează un singur fişier de configurare, plasaţi toate aceste directive în fişierul `httpd.conf`. Dacă versiunea dumneavoastră de Apache foloseşte abordarea mai veche cu trei fişiere pentru informaţia de configurare, plasaţi directiva `Alias` în fişierul `srm.conf`, respectiv liniile `Location` în fişierul `access.conf`.

Nu activaţi directivele `mod_perl`, `PerlSendHeader` sau `PerlSetupEnv` pentru catalogul `cgi-perl`. Acestea sunt manipulate automat de CGI.pm, iar activarea lor poate induce conflicte de prelucrare.

Adresa URL a unui script `mod-perl` este similară cu aceea a unui script CGI standard. Singura diferenţă este aceea că specificaţi `cgi-perl` în loc de `cgi-bin`:

```
http://numele.gazdei.dv/cgi-perl/nume_script
```

Pentru mai multe informaţii, vizitaţi regiunea `mod_perl` a sitului Web Apache, de la următoarea adresă:

```
http://perl/apache/org/
```

O scurtă introducere în CGI.pm

Pentru a scrie un script Perl care foloseşte modulul CGI.pm, plasaţi o linie `use` lângă începutul scriptului, apoi creaţi un obiect `CGI` care vă oferă acces la metodele şi variabilele modulului CGI.pm:

```
use CGI;
my ($cgi) = new CGI;
```

Scripturile noastre CGI folosesc caracteristicile modulului CGI.pm prin invocarea metodelor utilizând variabila `$cgi`. De exemplu, pentru a genera un titlu de nivel 1, vom folosi metoda `h1()` astfel:

```
print $cgi->h1 ("Titlul meu");
```

De asemenea, CGI.pm acceptă un stil de utilizare care vă permite să-i apelați metodele ca funcții, fără construcția inițială `$cgi->`. Nu folosesc aici sintaxa respectivă, deoarece notația `$cgi->` este mai asemănătoare cu modul de utilizare a modulului DBI, precum și deoarece împiedică apariția unor conflicte între numele funcțiilor din modulul CGI.pm și numele oricăror altor funcții pe care le definiți.

Verificarea parametrilor de intrare și scrierea datelor de ieșire

Una dintre operațiile pe care CGI.pm le execută automat este tratarea tuturor detaliilor „urâte" implicate în colectarea informațiilor de intrare furnizate de serverul Web scriptului dumneavoastră. Pentru a obține acele informații, tot ce aveți de făcut este să invocați metoda `param()`. Puteți obține numele tuturor parametrilor disponibili astfel:

```
my (@param) = $cgi->param ();
```

Pentru a regăsi valoarea unui anumit parametru, denumiți parametrul care vă interesează:

```
if (!$cgi->param ("parametrul_meu"))
{
   print "parametrul_meu nu este configurat\n";
}
else
{
   printf "Valoarea parametrului parametrul_meu: %s\n",
            $cgi->param ("parametrul_meu");
}
```

De asemenea, CGI.pm furnizează metode pentru generarea datelor de ieșire care trebuie trimise browserului client. Să considerăm următorul document HTML:

```
<HTML>
<HEAD>
<TITLE>Titlul paginii mele</TITLE>
</HEAD>
<BODY>
<H1>Titlu de nivel 1</H1>
<P>Paragraf 1.
<P>Paragraf 2.
</BODY>
</HTML>
```

Acest program folosește `$cgi` pentru a produce un document echivalent:

```
print $cgi->header ();
print $cgi->start_html (-title => "Titlul paginii mele");
print $cgi->h1 ("Antetul paginii mele");
print $cgi->p ();
print "Paragraf 1.\n";
print $cgi->p ();
print "Paragraf 2.\n";
print $cgi->end_html ();
```


Unul dintre avantajele utilizării modulului CGI.pm pentru generarea datelor de ieșire în loc de a scrie personal cod HTML brut sunt acelea că puteți gândi în unități logice, nu folosind etichete individuale de marcare, iar codul dumneavoastră HTML are mai puține șanse de a conține erori. (Motivul pentru care am spus că „are mai puține șanse" este acela că CGI.pm nu vă va împiedica să faceți lucruri bizare, cum ar fi includerea unei liste într-un titlu.) De asemenea, pentru textele (altele decât etichetele) pe care le scrieți, CGI.pm furnizează modificarea automată a semnificației unor caractere (escaping) precum < sau >, care sunt speciale pentru HTML.

Utilizarea metodelor modulului CGI.pm de generare a datelor de ieșire nu vă împiedică să scrieți personal cod HTML brut, dacă doriți. Puteți combina cele două metode, combinând apelurile la metodele CGI.pm cu instrucțiuni de afișare care generează etichete literale.

Modificarea automată a semnificației textelor HTML și URL

Dacă scrieți texte – diferite de textele etichetelor – cu ajutorul metodelor CGI.pm, precum `start_html()` sau `h1()`, programul modifică automat semnificația caracterelor speciale din text. De exemplu, dacă generați un titlu folosind următoarea instrucțiune, caracterul `&` din textul titlului va fi convertit în `&` de către CGI.pm:

```
print $cgi->start_html (-title => "A, B & C");
```

Dacă scrieți texte – altele decât textele etichetelor – fără a folosi metoda modulului CGI.pm de generare a datelor de ieșire, probabil că trebuie să le treceți mai întâi prin funcția `escapeHTML()`, pentru a vă asigura că semnificația tuturor caracterelor speciale a fost modificată în mod corespunzător. Acest fapt este de asemenea adevărat atunci când construiți adrese URL care pot conține caractere speciale, deși, în acest caz, trebuie să folosiți în schimb metoda `escape()`. Este important să folosiți metoda de codare adecvată, deoarece fiecare metodă tratează diferite seturi de caractere drept caractere speciale și codifică acele caractere speciale folosind formate care diferă unele de altele. Să luăm în considerare următorul script Perl scurt:

```
#! /usr/bin/perl
use CGI;
$cgi = new CGI;

$s = "x<= y, corect?";
print $cgi->escapeHTML ($s) ."\n";      # codificare pentru
                                        # utilizare ca text HTML
print $cgi->escape ($s) ."\n";          # codificare pentru
                                        # utilizare intr-un URL
```

Dacă rulați acest script, va produce următoarele date de ieșire, de unde puteți observa că procedeul de codificare pentru textul HTML nu este același ca în cazul adreselor URL:

```
x&lt;=y, corect?
x%3C%3Dy%2C%20corect%3F
```

Scrierea paginilor cu utilizări multiple

Unul din principalele motive pentru scrierea scripturilor bazate pe Web care generează cod HTML, în loc de a scrie documente HTML statice, este acela că un script poate produce diferite categorii de pagini, în funcție de modul în care este invocat. Toate scripturile CGI pe care le vom scrie au această proprietate. Fiecare din aceste scripturi funcționează astfel:

1. Când solicitați pentru prima dată scriptul de la browserul dumneavoastră, acesta generează o pagină inițială, care vă permite să selectați tipul de informație dorit.
2. Când efectuați o selecție, scriptul este re-invocat, dar de data aceasta regăsește și afișează într-o a doua pagină informațiile specifice pe care le-ați solicitat.

Principala problemă, în acest caz, este că doriți ca selecția pe care o efectuați din prima pagină să determine conținutul celei de-a doua pagini, dar, în mod normal, paginile Web sunt independente unele de altele, dacă nu realizați unele aranjamente speciale. Trucul este de a determina scriptul să genereze pagini care atribuie unui parametru o valoare care indică următoarei invocări a scriptului ceea ce doriți. Când invocați scriptul pentru prima dată, parametrul nu are nici o valoare; acest fapt indică scriptului să prezinte pagina sa inițială. Când indicați informațiile pe care doriți să le vedeți, pagina invocă din nou scriptul, dar cu parametrul configurat la o valoare care indică scriptului ce are de făcut.

Există diferite moduri de transfer al instrucțiunilor de la o pagină înapoi la un script. O modalitate este de a furniza utilizatorului un formular pe care acesta să-l completeze. Când utilizatorul trimite formularul, conținutul formularului este trimis serverului Web. Serverul transmite informațiile scriptului, care poate afla datele trimise de utilizator invocând metoda `param()`. Astfel vom proceda pentru cel de-al treilea script CGI al nostru (cel care permite utilizatorului să introducă un cuvânt cheie pentru a căuta în catalogul Ligii istorice).

Un alt mod de specificare a instrucțiunilor către un script este de a transfera informațiile ca parte a adresei URL pe care o trimiteți serverului Web atunci când solicitați scriptul. Astfel vom proceda pentru scripturile de navigare în tabelele din baza de date `samp_db`, respectiv de navigare prin tabelele cu punctaje. Această metodă se bazează pe faptul că scriptul generează o pagină care conține hiperlegături. Prin selectarea unei legături se invocă din nou scriptul, dar se specifică o valoare a parametrului care indică scriptului ce are de făcut. În schimb, scriptul se invocă singur în diferite moduri, pentru a furniza diferite categorii de rezultate, în funcție de legătura pe care o selectează utilizatorul.

Un script își poate permite să se auto-invoce expediind browserului o pagină care conține o hiperlegătură spre propriul său URL. De exemplu, un script denumit `scriptul_meu` poate scrie o pagină care să conțină această legătură:

```
<A HREF="/cgi-bin/scriptul_meu">Clic pe mine!</A>
```

Când utilizatorul execută clic pe textul „`Clic pe mine!`“, browserul utilizatorului trimite o cerere pentru `scriptul_meu` înapoi la serverul Web. Desigur, această acțiune nu va face decât să determine scriptul să trimită din nou aceeași pagină, deoarece nu au fost furnizate alte informații. Totuși, dacă atașați un parametru la URL, acel parametru este trimis înapoi serverului Web atunci când utilizatorul selectează legătura. Serverul invocă scriptul, iar scriptul poate apela funcția `param()` pentru a detecta că parametrul a fost configurat și poate întreprinde o acțiune în funcție de valoarea parametrului.

Pentru a atașa un parametru la sfârșitul adresei URL, adăugați un caracter `?` urmat de o pereche nume-valoare. Pentru a atașa mai mulți parametri, separați-i prin caractere `&`. De exemplu:

```
/cgi-bin/scriptul_meu?nume=valoare
/cgi-bin/scriptul_meu?nume=valoare&nume2=valoare2
```

Pentru a construi o adresă URL cu parametri atașați care face referire la sine însăși, un script CGI trebuie să înceapă prin a apela metoda `script_name()` pentru a-și obține propriul URL, iar apoi trebuie să atașeze parametri la adresă, astfel:

```
$url = $cgi->script_name ();          # obtine URL pentru script
$url .= "?nume=valoare";              # adauga primul parametru
$url .= "&nume2=valoare2";            # adauga al doilea parametru
```

După ce adresa URL a fost construită, puteți genera o etichetă de hiperlegătură `<A>` care conține adresa, folosind metoda `a()` a modulului CGI.pm:

```
print $cgi->a ({-href => $url}, "Clic pe mine!");
```

Pentru a vedea mai limpede care este modul de funcționare al acestei metode, vom examina un scurt script CGI. Când este invocat pentru prima dată, scriptul următor, denumit `flip_flop`, prezintă o pagină denumită Pagina A, care conține o singură hiperlegătură. Prin selectarea legăturii scriptul va fi invocat din nou, dar parametrul `pagina` este astfel configurat încât îi indică să afișeze pagina B. Și pagina B conține o legătură cu scriptul, dar fără nici o valoare pentru parametrul `pagina`. Ca atare, selectarea legăturii în pagina B determină afișarea din nou a paginii inițiale. Invocările ulterioare ale scriptului determină afișarea alternativă a paginilor A și B.

```
use CGI;

my ($cgi) = new CGI;
my ($url) = $cgi->script_name ();        # propriul URL al acestui script

print $cgi->header ();
if ($cgi->param ("pagina") ne "b")       # afiseaza pagina A
{
    print $cgi->start_html (-title => "Flip-Flop: Pagina A");
    print "Aceasta este pagina A.<BR>Pentru a selecta pagina B, ";
    $url .= "?pagina=b";                 # ataseaza parametrul
                                         # pentru a selecta pagina B
    print $cgi->a ({-href => $url}, "clic aici");
}
else                                     # afiseaza pagina B
{
    print $cgi->start_html (-title => "Flip-Flop: Pagina B");
    print "Aceasta este pagina B.<BR>Pentru a selecta pagina A, ";
    print $cgi->a ({-href => $url}, "clic aici");
}
print $cgi->end_html();
```

Dacă apare încă un client care solicită scriptul `flip_flop`, va fi prezentată pagina inițială, deoarece diferitele browsere ale clienților nu interacționează unele cu altele.

Valoarea variabilei `$url` a fost configurată într-o manieră destul de „cavalerească" în exemplele precedente. Este preferabil să folosiți metoda `escape()` pentru a codifica numele și valorile parametrilor dumneavoastră atunci când le atașați la un URL, în cazul în care conțin caractere speciale. Iată o modalitate mai bună de a construi un URL cu parametri atașați:

```
$url = $cgi->script_name ();              # obtine URL pentru script
$url .= sprintf ("?%s=%s",                # adauga primul parametru
      $cgi->escape ("nume"), $cgi->escape ("valoare"));
$url .= sprintf ("&%s=%s",                # adauga al doilea parametru
      $cgi->escape ("nume2"), $cgi->escape ("valoare2"));
```

Conectarea la serverul MySQL din scripturile Web

Scripturile în linie de comandă pe care le-am prezentat în secțiunea anterioară, „Utilizarea DBI", foloseau un preambul identic pentru stabilirea unei conexiuni cu serverul MySQL. Scripturile noastre CGI partajează și ele o anumită parte de program, însă puțin diferită:

```
use DBI;
use CGI;
use strict;

# Toti parametrii de conexiune prestabiliti lipsesc
my ($host_name, $user_name, $password) = (undef, undef, undef);
my (db_name) = "samp_db";

# construieste sursa de date
my ($dsn) = "DBI:mysql:$db_name";
$dsn .= ":hostname=$host_name" if $host_name;
$dsn .= ";mysql_read_default_file=/usr/local/apache/conf/samp_db.cnf";

# conectare la server
my (%attr) = ( RaiseError = 1 );
my ($dbh) = DBI->connect ($dsn, $user_name, $password, \%attr);
```

Acest preambul diferă de cel folosit pentru scripturile în linie de comandă din următoarele motive:

- Acum, prima secțiune conține o instrucțiune `use CGI`.
- Nu mai analizăm argumentele din linia de comandă.
- Programul caută parametrii de conexiune într-un fișier cu opțiuni, dar nu folosește fișierul `.my.cnf` din catalogul de bază al utilizatorului care rulează scriptul (cu alte cuvinte, catalogul de bază al utilizatorului serverului Web). Serverul Web poate rula scripturi pentru obținerea accesului la alte baze de date și nu avem nici un motiv să presupunem că toate acele scripturi vor folosi aceiași parametri de conexiune. În schimb, vom examina fișierul cu opțiuni stocat într-o altă locație (`/usr/local/apache/conf/samp_db.cnf`). Dacă doriți să folosiți un alt fișier, modificați calea de acces la fișierul cu opțiuni.

Scripturile invocate printr-un server Web rulează ca utilizator al serverului Web, nu ca dumneavoastră personal. Acest fapt ridică unele probleme de securitate, deoarece dumneavoastră pierdeți controlul după ce acesta este preluat de serverul Web. Trebuie să atribuiți proprietatea asupra fișierului cu opțiuni utilizatorului în numele căruia rulează serverul Web (poate `www`, `nimeni` sau altcineva similar) și să configurați modul la valoarea 400 sau 600, astfel încât nici un alt utilizator să nu-l mai poată citi. Din păcate, fișierul poate fi totuși citit de orice persoană care poate instala un script ce va fi executat de serverul Web. Tot ce are de făcut respectivul este să scrie un script care deschide în mod explicit fișierul cu opțiuni și îi afișează conținutul într-o pagină Web. Deoarece scriptul persoanei respective rulează ca utilizator al serverului Web, va avea permisiuni complete de citire a fișierului.

Din acest motiv, este prudent să creați un utilizator MySQL care are privilegii numai de citire (`SELECT`) asupra bazei de date `samp_db`, apoi menționați în fișierul `samp_db.cnf` numele și parola utilizatorului respectiv, nu pe acelea ale dumneavoastră. Astfel, nu riscați să permiteți scripturilor să se conecteze la baza dumneavoastră de date ca utilizator care are permisiunea de a modifica tabelele din baza de date respectivă. Capitolul 11, „Administrarea generală a sistemului MySQL", expune modul de creare a unui cont de utilizator MySQL cu privilegii limitate.

Alternativ, puteți aranja ca execuția scripturilor să aibă loc prin mecanismul suEXEC din Apache. Acesta vă permite să executați un script ca un anumit utilizator de încredere, după care scrie scriptul pentru a obține parametrii de conexiune dintr-un fișier cu opțiuni care poate fi citit numai de către utilizatorul respectiv. Puteți proceda astfel cu scripturile care necesită acces de scriere la baza de date, de exemplu.

O altă metodă constă în a scrie un script care să solicite un nume de utilizator și o parolă de la utilizatorul clientului, respectiv să folosească aceste valori pentru a stabili o conexiune cu serverul MySQL. Această soluție este mai adecvată pentru scripturi pe care le creați în scopuri administrative decât pentru scripturile de uz general. În orice caz, rețineți că unele metode de solicitare a numelui și a parolei pot fi atacate de orice persoană care poate insera un program de spionaj (în original *sniffer* – N.T.) în rețea între dumneavoastră și server.

Așa cum v-ați dat seama din paragrafele anterioare, securitatea scripturilor Web poate fi un aspect delicat. Categoric, constituie un subiect despre care trebuie să citiți mai mult, deoarece este un subiect de mare anvergură, care nu poate fi tratat aici după cum o merită. Puteți începe cu materialul dedicat securității din manualul Apache. De asemenea, este utilă secțiunea cu întrebări frecvente (FAQ) despre Web, de la adresa:

`http://www.w3.org/Security/Faq/`

Un browser pentru baza de date samp_db

Pentru prima noastră aplicație bazată pe Web, vom crea un script simplu – denumit `samp_browse` – care vă permite să vedeți care sunt tabelele din baza de date `samp_db`, precum și să examinați în mod interactiv conținutul acestor tabele din browserul dumneavoastră de Web. Scriptul `samp_browse` funcționează astfel:

- Când solicitați pentru prima dată `samp_browse` de la browserul dumneavoastră, acesta se conectează la serverul MySQL, regăsește o listă cu tabelele din baza de date `samp_db` și trimite browserului o pagină în care fiecare tabel este prezentat sub forma unei legături pe care o puteți selecta. Când selectați numele unui tabel din această pagină, browserul dumneavoastră trimite o cerere serverului Web, solicitând scriptului `samp_browse` să afișeze conținutul acelui tabel.
- Dacă `samp_browse` primește un nume de tabel de la serverul Web atunci când este invocat, regăsește conținutul tabelului respectiv și prezintă informațiile browserului dumneavoastră Web. Titlul fiecărei coloane de date este numele coloanei din tabel. Titlurile sunt prezentate sub formă de legături; dacă selectați una din ele, browserul dumneavoastră trimite o cerere serverului Web pentru a afișa același tabel, dar tabelul va fi acum sortat în funcție de coloana pe care ați selectat-o.

Un avertisment: tabelele din baza de date `samp_db` sunt relativ mici, deci trimiterea întregului conținut al unui tabel către browserul dumneavoastră nu reprezintă o problemă. Dacă editați `samp_db` astfel încât să afișeze tabele dintr-o altă bază de date, care conține tabele mari, trebuie să luați în considerare adăugarea unei clauze `LIMIT` la instrucțiunile de regăsire a rândurilor.

În corpul principal al scriptului `samp_browse`, creăm obiectul CGI și scriem partea inițială a paginii Web. Apoi, verificăm dacă trebuie sau nu să afișăm un anumit tabel, în funcție de valoarea parametrului `nume_tabel`:

```
my ($cgi) = new CGI;

# scrie partea initiala a paginii
my ($titlu) = "$db_name browser de baze de date";
print $cgi->header ();
print $cgi->start_html (-title => $titlu);
print $cgi->h1 ($title);

# parametri de cautat in URL
my ($nume_tabel) = $cgi->param ("nume_tabel");
my ($coloana_sortare) = $cgi->param ("coloana_sortare");

# daca $nume_tabel nu are nici o valoare, afiseaza o lista
# cu tabele pe care se poate executa clic. Altfel, afiseaza
# continutul tabelului dat. Parametrul $nume_coloana, daca
# are o valoare, arata coloana in functie de care se va sorta.
if (!$nume_tabel)
```



```
{
   afiseaza_lista_tabele()
}
else
{
   afiseaza_tabel ($nume_tabel, $coloana_sortare);
}
print $cgi->end_html ();
```

Este ușor de aflat valoarea unui parametru, deoarece CGI.pm execută toate operațiile de determinare a informațiilor pe care serverul Web le transmite scriptului. Trebuie numai să apelăm funcția `param()` cu numele parametrului care ne interesează. În corpul principal al scriptului `samp_browse`, parametrul respectiv este `nume_tabel`. Dacă parametrul nu este definit sau este vid, aceasta este invocarea inițială a scriptului și se afișează lista cu tabele. În caz contrar, se afișează conținutul tabelului desemnat de parametrul `nume_tabel`, sortat în funcție de valorile din coloana determinată de parametrul `coloana_sortare`. După afișarea informațiilor adecvate, apelăm funcția `end_html()` pentru a insera etichetele HTML de închidere.

Funcția `afiseaza_lista_tabele()` generează pagina inițială. Funcția `afiseaza_lista_tabele()` regăsește lista cu tabele și scrie un tabel HTML cu o singură coloană, care conține în fiecare celulă numele unui tabel din baza de date:

```
sub afiseaza_lista_tabele
{
my ($ary_ref, $url);

   print "Selectati un tabel executand clic pe numele lui:<BR><BR>\n";

   # regaseste referinta la un tablou dintr-o singura coloana,
   # alcatuit din numele tabelelor
   $ary_ref = $dbh->selectcol_arrayref (qq{ SHOW TABLES FROM $db_name });

   # afiseaza tabelul cu un chenar
   print "<TABLE BORDER>\n";
   print "<TR>\n";
   afiseaza_celula ("TH", "Numele tabelului", 1);
   print "</TR>\n";
   foreach my $nume_tabel (@{$ary_ref})
   {
      $url = $cgi->script_name ();
      $url .= sprintf ("?nume_tabel=%s", $cgi->escape ($nume_tabel));
      print "<TR>\n";
      afiseaza_celula ("TD", $cgi->a ({-href => $url}, $nume_tabel), 0);
      print "</TR>\n";
   }
   print "</TABLE>\n";
}
```

Pagina generată de funcția `afiseaza_lista_tabele()` conține legături care se prezintă astfel:

```
/cgi-bin/samp_browse?nume_tabel = absente
/cgi-bin/samp_browse?nume_tabel = eveniment
/cgi-bin/samp_browse?nume_tabel = membru
...
```

Dacă parametrul `nume_tabel` are o valoare atunci când este invocat scriptul `samp_browse`, scriptul transferă valoarea funcției `afiseaza_tabel()`, alături de numele coloanei în funcție de care se vor sorta rezultatele. Dacă nu este specificată nici o coloană, sortarea se va efectua în funcție de prima coloană (putem face referire la coloane după poziția acestora, deci această operație se poate executa ușor folosind o clauză `ORDER BY 1`):

```
sub afiseaza_tabel()
{
my ($nume_tabel, $coloana_sortare) = @_;
my ($sth, $url);

    # daca nu este specificata coloana de sortare,
    # foloseste prima coloana
    $coloana_sortare = "1" unless $coloana_sortare;

    # prezinta o legatura care readuce utilizatorul la
    # pagina cu lista tabelelor
    print $cgi->a ({-href => $cgi->script_name ()}, "Afiseaza lista cu tabele");
    print "<BR><BR>\n";

    $sth = $dbh->prepare (qq{
            SELECT * FROM $nume_tabel ORDER BY $coloana_sortare
            });
    $sth->execute();

    print "<B> Continutul tabelului $nume_tabel:</B><BR>\n";

    # afiseaza tabelul cu un chenar
    print "<TABLE BORDER>\n";
    # foloseste numele coloanelor pentru capetele de tabel;
    # transforma fiecare capat de tabel intr-o legatura, care
    # sorteaza datele in functie de coloana corespunzatoare
    print "<TR>\n";
    foreach my $nume_coloana (@{$sth->{NAME}})
    {
      $url = $cgi->script_name ();
      $url .= sprintf ("?nume_tabel=%s", $cgi->escape ($nume_tabel));
      $url .= sprintf ("&coloana_sortare=%s", $cgi->escape ($nume_coloana));
      afiseaza_celula ("TH", $cgi->a ({-href => $url}, $nume_coloana), 0);
    }
    print "</TR>\n";

    # afiseaza randurile din tabel
    while (my @ary = $sth->fetchrow_array())
    {
      print "<TR>\n";
      foreach my $val (@ary)
      {
            afiseaza_celula ("TD", $val, 1);
      }
      print "</TR>\n";
    }

    $sth->finish ();
    print "</TABLE>\n";
}
```


O pagină de afișare a tabelelor asociază capetele de coloană cu legăturile care determină afișarea din nou a tabelului; aceste legături includ un parametru `$coloana_sortare`, care specifică în mod explicit coloana în funcție de care se va efectua sortarea. De exemplu, pentru o pagină care afișează conținutul tabelului `eveniment`, legăturile de tip cap de coloană se prezintă astfel:

```
/cgi-bin/samp_browse?nume_tabel = eveniment&coloana_sortare=data
/cgi-bin/samp_browse?nume_tabel = eveniment&coloana_sortare=tip
/cgi-bin/samp_browse?nume_tabel = eveniment&coloana_sortare=eveniment_id
```

Atât `afiseaza_lista_tabel()`, cât și `afiseaza_tabel()` folosesc funcția `afiseaza_celula()`, o funcție utilitară care afișează o valoare sub forma unei celule într-un tabel HTML. Această funcție folosește un mic truc, prin care transformă valorile vide într-un spațiu neîntreruptibil (` `), deoarece, într-un tabel cu chenar, celulele vide nu afișează chenarele în mod adecvat. Inserția în celulă a unui spațiu neîntreruptibil rezolvă problema. `afiseaza_celula()` mai preia un al treilea parametru, care controlează codificarea sau nu a valorii din celulă. Acest lucru este necesar deoarece `afiseaza_celula()` este apelată pentru a afișa unele valori din celule care au fost deja codificate, cum ar fi capetele de coloană care conțin informații despre URL.

```
# afiseaza o valoare in celula unui tabel; insereaza spatii
# neintreruptibile in celulele "vide" pentru a permite afisarea chenarelor

sub afiseaza_celula
{
my ($eticheta, $valoare, $codare) = @_;
```

Continuare

Continuare

```
    $valoare = $cgi->escapeHTML ($valoare) if $codare;
    $valoare = " " unless $valoare;
    print "<$eticheta$valoare/$eticheta>\n";
}
```

Dacă doriți să scrieți un script mai general, puteți modifica scriptul `samp_browse` astfel încât să parcurgă mai multe baze de date. De exemplu, scriptul poate începe prin a afișa o listă cu bazele de date din server, nu o listă cu tabele dintr-o anumită bază de date. Apoi, puteți alege o bază de date pentru a obține o listă a tabelelor sale și puteți continua începând din acest punct.

Browserul de parcurgere a punctajelor din cadrul proiectului de evidență a rezultatelor școlare

De fiecare dată când introducem punctajele obținute la un test, trebuie să generăm o listă ordonată cu punctaje, astfel încât să putem determina curba de notare și să putem atribui note literale. Rețineți că nu vom face decât să afișăm această listă, pentru a determina unde se găsește limita de acordare a fiecărei note literale. Apoi, vom trece notele pe lucrările elevilor înainte de a le înapoia acestora. Nu vom înregistra notele elevilor în baza de date, deoarece notele de la sfârșitul perioadei de examinare se bazează pe punctaje numerice, nu pe note literale. Mai rețineți că, într-un sens strict, trebuie să dispunem de o modalitate de a introduce punctajele înainte de a crea o metodă de a le regăsi. Am lăsat scriptul pentru introducerea punctajelor în capitolul următor. Între timp, deja avem în baza de date numeroase seturi de punctaje, încă din prima parte a perioadei de examinare. Ne putem folosi scriptul cu aceste punctaje, chiar în absența unei metode convenabile de introducere a punctajului.

Scriptul nostru de parcurgere a punctajelor, `score_browse`, prezintă unele asemănări cu `samp_browse`, dar este destinat scopului mai concret de examinare a punctajelor pentru un test sau pentru un chestionar dat. Pagina inițială prezintă o listă a posibilelor evenimente de examinare din care se poate alege și permite utilizatorului să îl selecteze pe oricare dintre acestea, pentru a vedea punctajele asociate evenimentului. Punctajele pentru un eveniment dat sunt sortate în funcție de valoare, în ordine descrescătoare, deci puteți afișa rezultatul și îl puteți folosi pentru a determina curba de notare.

Scriptul `score_browse` trebuie să examineze un singur parametru, și anume `eveniment_id`, pentru a determina dacă a fost specificat sau nu un anumit eveniment. Dacă nu, `score_browse` afișează rândurile tabelului `eveniment`, astfel încât utilizatorul să poată selecta un eveniment. În caz contrar, afișează punctajele asociate evenimentului ales:

```
# parametrul care ne indica evenimentul pentru care
# trebuie afisate punctajele
my ($eveniment_id) = $cgi-param ("eveniment_id");

# daca eveniment_id are o valoare, afiseaza lista cu evenimente;
# altfel, afiseaza punctajele pentru evenimentul dat.
```



```
if (!$eveniment_id)
{
   afiseaza_evenimente ()
}
else
{
   afiseaza_punctaje ($eveniment_id);
}
```

Funcția `afiseaza_evenimente()` extrage informații din tabelul `eveniment` și le afișează în formă tabelară, folosind numele coloanelor din interogare drept capete de coloană pentru tabel. În cadrul fiecărui rând, valoarea `eveniment_id` este afișată sub forma unei legături care poate fi selectată pentru a declanșa o interogare care regăsește punctajele corespunzătoare evenimentului. Adresa URL pentru fiecare eveniment este pur și simplu calea de acces spre scriptul `score_browse`, la care este atașat un parametru ce specifică numărul evenimentului:

```
/cgi-bin/score_browse?eveniment_id=numar
```

Funcția `afiseaza_evenimente()` se scrie astfel:

```
sub afiseaza_evenimente
{
my ($sth, $url);
   print "Selectati un eveniment executand clic pe numarul sau:<BR><BR>\n";

   # obtine lista cu evenimente
   $sth = $dbh->prepare (qq{
      SELECT eveniment_id, data, tip
      FROM eveniment
      ORDER BY eveniment_id
   });
   $sth->execute ();

   # afiseaza tabelul cu un chenar
   print "<TABLE BORDER>\n";
   # foloseste numele coloanelor drept capete de coloana in tabel
   print "<TR>\n";
   foreach my $nume_coloana (@{$sth->{NAME}})
   {
      afiseaza_celula ("TH", $nume_coloana, 1);
   }
   print "</TR>\n";

   # asociaza fiecare identificator de eveniment cu o legatura
   # care va afisa punctajele de la acel eveniment; returneaza
   # randurile folosind un hash pentru a facilita referirea la
```

Continuare

Continuare

```
      # valoarea coloanei eveniment_id in functie de nume.
      while (my $hash_ref = $sth->fetchrow_hashref ())
      {
         print "<TR>\n";
         $url = $cgi->script_name();
         $url .= sprintf ("?eveniment_id=%s",
              $cgi->escape ($hash_ref->{eveniment_id}));
         afiseaza_celula ("TD", $cgi->a ({-href => $url},
              $hash_ref->{eveniment_id}), 0);
         afiseaza_celula ("TD", $hash_ref->{data}), 1);
         afiseaza_celula ("TD", $hash_ref->{tip}), 1);
         print "</TR>\n";
      }
      $sth->finish ();
      print ("</TABLE>\n")
   }
```

Când utilizatorul selectează un eveniment, browserul trimite o cerere pentru scriptul `score_browse`, la care este atașată o valoare a identificatorului de eveniment. `score_browse` găsește setul de parametri `eveniment_id` și apelează funcția `afiseaza_punctaje()` pentru a afișa toate punctajele obținute la evenimentul specificat. De asemenea, pagina afișează textul `"Afiseaza lista cu evenimente"` ca o legătură orientată înapoi la pagina inițială, astfel încât utilizatorul poate reveni cu ușurință la pagina care conține lista cu evenimente. Adresa URL pentru această legătură face referire la scriptul `score_browse`, dar nu specifică nici o valoare pentru parametrul `eveniment_id`. Funcția `afiseaza_punctaje()` este prezentată în listingul următor:

```
sub afiseaza_punctaje
{
my ($eveniment_id) = shift;
my ($sth);

   # un URL fara nici un parametru eveniment_id va determina
   # afisarea listei cu evenimente.
   print $cgi->a ({-href => $cgi->script_name ()},
      "Afiseaza lista cu evenimente");
   print "<BR><BR>\n";

   # selecteaza punctajele pentru evenimentul dat
   $sth = $dbh->prepare (qq{
      SELECT
           elev.nume, eveniment.data, puncte.puncte, eveniment.tip
      FROM
           elev, puncte, eveniment
      WHERE
           elev.elev_id = puncte.elev_id
           AND puncte.eveniment_id = eveniment.eveniment_id
           AND eveniment.eveniment_id = ?
      ORDER BY
           eveniment.data ASC, eveniment.tip ASC, puncte.puncte DESC
   }
   $sth->execute ($eveniment_id);      # transfera identificatorul
                                       # de eveniment ca valoare
                                       # pentru campul de inlocuire

   print "<B>Punctaje pentru evenimentul $eveniment_id/<B><BR>\n";

   # afiseaza tabelul cu un chenar
   print "<TABLE BORDER>\n";
   # utilizeaza numele coloanelor drept capete de coloana ale tabelului
   print "<TR>\n";
   foreach my $nume_coloana (@{$sth->{NAME}})
   {
      afiseaza_celula ("TH", $nume_coloana, 1);
   }
   print "</TR>\n";

   while (my @ary = $sth->fetchrow_array ())
   {
      print "<TR>\n";
      afiseaza_celula ("TD", shift (@ary), 1) while @ary;
      print "</TR>\n";
   }

   $sth->finish ();
   print "</TABLE>\n";
}
```


Interogarea pe care o rulează scriptul `afiseaza_punctaje()` este foarte similară celei pe care am creat-o cu mult timp în urmă, în capitolul 1, în secțiunea „Regăsirea informațiilor din mai multe tabele", care prezenta metode de scriere a unirilor. În acel capitol, am cerut punctajele obținute la o anumită dată, deoarece datele sunt mai semnificative decât valorile identificatorilor de eveniment. Dimpotrivă, când folosim `score_browse`, cunoaștem cu exactitate identificatorul de eveniment. Aceasta nu deoarece folosim ca reper identificatorul de eveniment (ceea ce nu facem), ci fiindcă scriptul ne prezintă o listă cu asemenea identificatori, din care trebuie să alegem. Se poate observa că acest tip de interfață reduce necesitatea de a cunoaște anumite detalii. Nu trebuie să cunoaștem un identificator de eveniment; trebuie numai să recunoaștem evenimentul de care avem nevoie.

Identificarea membrilor Ligii istorice care manifestă interese comune

Scripturile `samp_browse` și `score_browse` permit utilizatorului să opereze o selecție prin prezentarea unei liste cu opțiuni într-o pagină inițială, unde fiecare opțiune este o legătură care re-invocă scriptul, de data aceasta cu o anumită valoare a unui parametru. O altă modalitate de a permite utilizatorilor să opteze este de a insera într-o pagină un formular care conține un câmp ce poate fi editat. Acest procedeu este mai adecvat atunci când domeniul opțiunilor posibile nu este limitat la un set de valori ușor de determinat. Următorul nostru script demonstrează această metodă de a solicita date de intrare de la utilizator.

În secțiunea „Utilizarea DBI", am construit un script în linie de comandă pentru a găsi pe membrii Ligii istorice care manifestă un anumit interes. Totuși, membrii Ligii nu au acces la acest script; secretarul Ligii trebuie să ruleze scriptul și apoi să expedieze prin e-mail rezultatul membrului care a solicitat lista. Ar fi interesant ca această posibilitate de căutare să fie pusă la dispoziția mai multor membri, astfel încât aceștia să o poată folosi personal. Scrierea unui script Web este o modalitate de a rezolva problema.

Scriptul de față, intitulat `interese`, afișează un mic formular în care utilizatorul poate introduce un cuvânt cheie, iar apoi caută în tabelul `membru` pentru a găsi membrii corespunzători și afișează rezultatele. Căutarea se execută adăugând caracterul de înlocuire `%` (procent) la ambele extremități ale cuvântului cheie, astfel încât acesta să poată fi găsit oriunde în cadrul valorilor din coloana `interese`.

Formularul care folosește cuvântul cheie este afișat în fiecare pagină, astfel încât utilizatorul să poată introduce imediat o nouă căutare, chiar din acele pagini care afișează rezultatele căutării. În plus, șirul de căutare din pagina anterioară este afișat sub formă de cuvânt cheie, astfel încât, dacă utilizatorul dorește să ruleze o căutare similară, șirul să poată fi editat. Astfel, nu este necesară reintroducerea unei mari cantități de text:

```
# parametrul de cautat
my ($interes) = $cgi->param ("interes");

# Afiseaza un formular pentru introducerea cuvintelor cheie. In plus, daca
# variabila $interes este definita, cauta si afiseaza o lista a membrilor
# care manifesta acel interes. Observati ca valoarea curenta a variabilei
# $interes este afisata ca valoare prestabilita a campului "interes"
# din formular.

print $cgi->start_form (-method => "POST");
print $cgi->textfield (       -name => "interes",
                              -value => $interes,
                              -size => 40);
print $cgi->submit ( -name => "buton", -value => "Cauta");
print $cgi->end_form ();

# ruleaza o cautare daca a fost specificat un cuvant cheie
cauta_membri ($interes) if $interes;
```


Scriptul își comunică lui însuși informațiile într-un mod ușor diferit față de `samp_browse` sau `score_browse`. Parametrul `interes` nu este adăugat la sfârșitul unui URL. În schimb, informațiile din formular sunt codificate de către browser și trimise ca parte a unei cereri POST. Totuși, pentru `CGI.pm` nu are nici o importanță modul de transmitere a informației; valoarea parametrului este obținută tot prin apelarea funcției `param()`.

Funcția pentru executarea căutării și pentru afișarea rezultatelor este prezentată în listingul următor. Funcția care formatează intrarea, `format_html_entry()`, nu a fost prezentată, deoarece este similară celei din scriptul `gen_cat`:

```
sub cauta_membri
{
my ($interes) = shift;
my ($sth, $numar);

   printf "Rezultatele cautarii pentru cuvantul cheie:
           %s<BR><BR>\n", $cgi->escape_html ($interes);
   $sth = $dbh->prepare (qq{
           SELECT * FROM membru WHERE interese LIKE ?
           ORDER BY nume, prenume
      });
   # cauta sirul oriunde in campul "interese"
   $sth->execute ("%" . $interes . "%");
   $numar = 0;
   while (my $hash_ref = $sth->fetchrow_hashref ())
   {
      format_html_entry ($hash_ref);
      ++$numar;
   }
   print $cgi->p ("Au fost gasite $numar intrari");
}
```

CAPITOLUL 8

Interfața API pentru PHP

PHP este un limbaj de scripting ce vă permite să scrieți pagini Web care conțin programe înglobate, programe ce se execută la fiecare deschidere a paginii și care pot genera un conținut dinamic care va fi inclus ca parte a datelor de ieșire trimise browserului Web al unui client. Acest capitol descrie modul de utilizare a limbajului PHP pentru a scrie aplicații bazate pe Web care folosesc MySQL. Pentru o comparație între PHP și interfețele API pentru programarea în MySQL scrise în C, respectiv Perl DBI, consultați capitolul 5, „Introducere în programarea MySQL".

Exemplele din capitolul de față pornesc de la baza noastră de date demonstrativă, `samp_db`, folosind tabelele pe care le-am creat pentru proiectul de evidență a rezultatelor școlare și pentru Liga istorică în capitolul 1, „Introducere în MySQL și SQL". Vom discuta despre PHP 3, deși PHP 4 se află deja la versiunea beta în momentul scrierii acestei cărți și poate că va deveni deja disponibil până când veți ajunge să citiți rândurile de față. Compatibilitatea cu PHP 3 este unul dintre scopurile de proiectare specificate cu claritate ale versiunii PHP 4, deci aproape toate cele prezentate aici pentru PHP 3 sunt valabile și pentru PHP 4. Un set de note pentru trecerea la PHP 4 descrie modificările în raport cu PHP 3. Dacă folosiți PHP 4, trebuie să citiți aceste note.

Acest capitol a fost scris pornind de la presupunerea că veți folosi PHP în conjuncție cu serverul Web Apache. Biblioteca client pentru MySQL scrisă în C și fișierele antet trebuie să fie de asemenea instalate; aceste fișiere sunt necesare atunci când construiți PHP; în caz contrar, PHP nu va ști cum să obțină accesul la bazele de date MySQL. Dacă trebuie să vă procurați oricare dintre aceste programe, consultați Anexa A, „Obținerea și instalarea programelor". Instrucțiunile pentru obținerea scripturilor date ca exemplu, create în acest capitol, se găsesc de asemenea în anexa respectivă. Puteți descărca scripturile pentru a evita să le introduceți personal de la tastatură.

Sub UNIX, PHP se poate folosi cu Apache fie ca modul încorporat care este legat în fișierul binar executabil Apache, fie ca interpretor autonom, folosit ca program CGI tradițional. Sub Windows, PHP poate rula numai ca program autonom în momentul de față, deși se lucrează la dezvoltarea unui modul PHP 4 Apache, care va rula sub Windows NT.

În majoritatea cazurilor, acest capitol descrie funcțiile PHP numai în măsura în care sunt necesare în expunere. Pentru un listing mai comprehensiv al tuturor funcțiilor legate de MySQL, vezi Anexa H, „Referință API PHP". De asemenea, probabil că veți dori să consultați manualul PHP, care descrie gama completă de funcții pe care le furnizează PHP, inclusiv funcțiile pentru utilizarea sistemelor de baze de date diferite de MySQL.

(PHP, ca și DBI, nu este constrâns să funcționeze numai cu MySQL.) Manualul este disponibil din situl Web PHP, la adresa `http://www.php.net/`. De asemenea, acest sit Web conține notele pentru trecerea de la PHP 3 la PHP 4.

Caracteristicile scripturilor PHP

Numele fișierelor care conțin scripturile PHP sunt scrise, în mod caracteristic, cu o extensie care permite serverului dumneavoastră de Web să le recunoască și să execute interpretorul PHP pentru a le prelucra. Dacă folosiți o extensie care nu este recunoscută, scripturile dumneavoastră PHP vor fi servite în format text simplu. Extensia folosită în acest capitol este `.php`. Alte extensii frecvent folosite sunt `.php3` și `.phtml`. Pentru instrucțiuni de configurare a serverului Apache astfel încât să recunoască extensia pe care doriți să o folosiți, consultați Anexa A. Dacă nu puteți controla instalarea serverului Apache pe calculatorul dumneavoastră, aflați de la administratorul de sistem care este extensia adecvată pentru a fi utilizată.

Noțiuni fundamentale despre PHP

Funcția elementară a limbajului PHP este de a interpreta un script pentru a genera o pagină Web care este trimisă unui client. În mod caracteristic, scriptul conține o combinație între coduri HTML, care sunt trimise literal clientului, și linii de program PHP, care sunt executate ca program. Datele de ieșire produse de program sunt trimise clientului, indiferent de natura lor, deci clientul nu vede niciodată programul, ci numai datele de ieșire rezultante.

Când PHP începe să citească un fișier, pur și simplu copiază tot ceea ce găsește acolo în datele de ieșire, cu presupunerea că fișierul conține HTML literal. Când interpretorul PHP întâlnește o etichetă specială de deschidere, comută din modul HTML în modul program PHP și începe să interpreteze fișierul drept linii de program PHP care trebuie executate. Sfârșitul programului este indicat de o altă etichetă specială, moment în care interpretorul comută înapoi din modul program în modul HTML. Aceasta vă permite să combinați un text static (partea de HTML) cu rezultate generate dinamic (date de ieșire ale programului din partea de PHP) pentru a produce o pagină care variază în funcție de circumstanțele în care este apelată.

De exemplu, puteți folosi un script PHP pentru a prelucra rezultatul unui formular în care un utilizator a introdus parametri pentru o căutare într-o bază de date. Parametrii de căutare pot fi diferiți pentru variante diferite de completare a formularului, deci, atunci când scriptul execută căutări, fiecare pagină rezultantă va reflecta o altă căutare.

Să vedem cum funcționează aceste procedeu, examinând un script PHP extrem de simplu:

```
<HTML>
<BODY>
Salut, lume
</BODY>
</HTML>
```

Acest script nu este foarte interesant, deoarece nu conține nici o linie de program PHP! La ce bun? veți întreba. Aceasta este o întrebare logică. Răspunsul este că uneori este utilă configurarea unui script care conține numai cadrul HTML pentru pagina pe care doriți să o generați, care apoi va fi adăugat la liniile de program PHP. Este un procedeu absolut corect și interpretorul PHP nu are nici o obiecție.

Pentru a include linii de program PHP într-un script, acestea se deosebesc de textul înconjurător cu ajutorul a două etichete speciale, în speță ?php pentru a începe scriptul, respectiv ? pentru a-l încheia. Când interpretorul PHP întâlnește eticheta de deschidere ?php, comută din modul HTML în modul program PHP și interpretează tot ceea ce găsește drept linii de program PHP până când vede eticheta de închidere ?. Scriptul cuprins între cele două etichete este interpretat și înlocuit cu datele de ieșire pe care le generează. Exemplul anterior poate fi scris din nou, pentru a include un mic exemplu de program PHP, astfel:

```
<HTML>
<BODY>
<?php print ("Salut, lume\n"); ?>
</BODY>
</HTML>
```

În acest caz, partea de program este redusă la minimum, fiind alcătuită dintr-o singură linie. Când programul este interpretat, generează rezultatul `Salut, lume`, care devine parte a datelor de ieșire trimise browserului clientului. Astfel, pagina Web produsă de acest script este similară celei produse în exemplul anterior, unde scriptul era alcătuit în totalitate din linii HTML.

Puteți folosi linii de program PHP pentru a genera orice componentă a unei pagini Web. Deja am văzut una dintre extreme, în care întregul script era alcătuit din HTML literal și nu conținea nici o linie de program PHP. Cealaltă extremă este un script alcătuit în totalitate din linii de program PHP și care nu conține HTML literal:

```
<?php
print ("<HTML>\n");
print ("<BODY>\n");
print ("Salut, lume\n");
print ("</BODY>\n");
print ("</HTML>\n");
?>
```

Acest fapt demonstrează că PHP vă oferă o mare flexibilitate în ceea ce privește generarea datelor de ieșire. PHP lasă la latitudinea dumneavoastră alegerea combinației celei mai adecvate între HTML și liniile de program PHP. O altă dovadă a flexibilității limbajului PHP constă în aceea că nu este necesar să inserați toate liniile de program într-un singur loc. Puteți comuta între modul HTML și modul program PHP pe tot parcursul scriptului, oricum și oricând doriți.

Etichetele scripturilor PHP

În afară de etichetele folosite pentru exemplele din acest capitol, PHP înțelege și alte etichete de script. Le puteți vedea în liniile de program PHP pe care le scriu alte persoane sau le puteți folosi personal. PHP recunoaște patru stiluri de etichete:

- Stilul de etichetă prestabilit. Acesta este stilul pe care PHP este configurat să-l folosească în mod prestabilit:

```
<?php print ("Salut, lume\n"); ?>
```

- Stil cu etichetă de deschidere scurtă. Acesta este asemănător stilului prestabilit, cu excepția faptului că eticheta de deschidere este mai scurtă:

```
<? print ("Salut, lume\n"); ?>
```

- Stil compatibil ASP. Acest stil este comun în mediile Active Server Pages:

```
<% print ("Salut, lume\n"); %>
```

- Stil de etichetă `SCRIPT`. Acest stil este util dacă folosiți un editor HTML care nu agreează nici unul din celelalte stiluri de etichete. Categoric se folosește mai mult text, dar stilul poate fi necesar când un editor nu vă acceptă programul PHP deoarece folosiți alte stiluri de etichete:

```
<SCRIPT LANGUAGE="php"> print ("Salut, lume\n"); </SCRIPT>
```

Stilul cu etichetă de deschidere scurtă și stilul compatibil ASP nu sunt activate în mod prestabilit. Pentru mai multe instrucțiuni pentru activarea acestor etichete, consultați Anexa H.

Scripturi PHP autonome

Este posibil să scrieți scripturi PHP autonome, pe care le puteți invoca din linia de comandă, așa cum procedați cu un script de interpretor sau cu un script Perl. Iată un exemplu:

```
#! /usr/local/bin/php -q
<?php print ("Salut, lume\n"); ?>
```

Scriptul anterior poate fi denumit `salut.php`, poate deveni executabil folosind comanda chmod +x și poate fi invocat din interpretor:

```
% salut.php
Salut, lume
```

În acest capitol nu vom scrie nici un script autonom. Toate exemplele au fost scrise pornind de la premisa că vor fi invocate de către un server Web pentru a genera o pagină Web.

Scriptul următor este ceva mai substanțial, dar relativ scurt, și prezintă modul simplu în care puteți obține acces la o bază de date MySQL din PHP, respectiv modul în care puteți folosi rezultatele unei interogări într-o pagină Web. Scriptul următor a fost prezentat foarte succint în capitolul 5 și reprezintă un fundament simplu al unei pagini de bază a sitului Ligii istorice. Pe parcurs, vom adăuga scriptului un plus de complexitate, dar, pentru moment, nu face decât să afișeze un scurt mesaj de salut și numărul actual al membrilor Ligii:

```
<HTML>
<HEAD>
<TITLE>Liga istorica americana</TITLE>
</HEAD>
```

Continuare

Continuare

```
<BODY>
<P>Bine ati venit la situl Web al Ligii istorice.
<?php
$link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
   or exit ();
mysql_select_db ("samp_db")
   or exit ();
$result = mysql_query ("SELECT COUNT(*) FROM membru")
   or exit ();
if ($row = mysql_fetch_array ($result))
   echo "<P>In prezent, Liga are " . $row[0] . " membri";
mysql_free_result ($result);
?>
</BODY></HTML>
```

Mesajul de întâmpinare este numai un text static, deci cel mai uşor este să-l scrieţi pur şi simplu sub formă de HTML literal. Pe de altă parte, numărul membrilor este dinamic şi se modifică în timp, deci trebuie determinat instantaneu prin interogarea tabelului `membru` din baza de date `samp_db`.

Textul programului delimitat de etichetele de deschidere şi de închidere a scriptului execută o operaţie simplă. Mai întâi, deschide o conexiune cu serverul MySQL şi transformă `samp_db` în bază de date prestabilită. Apoi, trimite o interogare serverului pentru a determina numărul de membri ai Ligii istorice din momentul respectiv (pe care îl evaluăm ca fiind numărul de rânduri din tabelul `membru`). Rezultatul interogării este afişat ca parte a unui mesaj care conţine numărul de membri, iar apoi este eliminat.

Dacă în orice moment al acestui proces se produce vreo eroare, scriptul îşi încheie pur şi simplu execuţia. Nu sunt generate nici un fel de date de ieşire privind eroarea produsă, deoarece acestea pot fi derutante pentru cei care vizitează situl Web.[1]

Să defalcăm scriptul pe componente, pentru a vedea care este modul de funcţionare al fiecăreia. Prima etapă constă în conectarea la server folosind funcţia `mysql_pconnect()`:

```
$link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
   or exit ();
```

Funcţia `mysql_pconnect()` preia ca argumente numele gazdei, numele utilizatorului şi parola şi returnează un identificator de legătură în cazul în care conexiunea a fost stabilită cu succes, respectiv `FALSE` dacă se produce vreo eroare. Dacă nu reuşeşte încercarea de conexiune, scriptul nostru apelează funcţia `exit()`, care termină imediat execuţia scriptului, caz în care nu mai sunt generate alte date de ieşire.

[1] Dacă generaţi întreaga pagină Web prin intermediul liniilor de program PHP, oprirea programului în caz de eroare fără afişarea nici unor date de ieşire îi poate agasa pe cei care vă citesc paginile, deoarece unele browsere vor afişa o casetă de dialog cu mesajul „această pagină nu conţine nici un fel de date", casetă care trebuie eliminată. – N.A.

Dar ce înseamnă caracterul `@` din faţa apelului la funcţia `mysql_pconnect()`? Acesta este caracterul „Linişte, vă rog". Unele funcţii PHP scriu un mesaj de eroare în caz de eşec, pe lângă returnarea unui cod de stare. În cazul funcţiei `mysql_pconnect()`, o încercare ratată de conexiune determină apariţia în pagina Web care este trimisă către browserul clientului a unui mesaj ca acesta:

```
Warning: MySQL Connection Failed: Access denied for user:
'paul@pit-viper.snake.net' (Using password: YES)²
```

Asta nu-i frumos; persoana care ne vizitează situl nu va şti cum să procedeze cu acest mesaj. Inserţia caracterului `@` în faţa apelului la funcţia `mysql_pconnect()` elimină acest mesaj de eroare, astfel încât să putem alege singuri modul de tratare a erorilor, pe baza valorii returnate. Pentru scriptul curent, cel mai bun lucru în cazul unei erori este de a nu genera nici un fel de date de ieşire legate de numărul membrilor. În acest caz, pagina va conţine numai mesajul de întâmpinare.

Puteţi folosi caracterul `@` cu orice funcţie PHP, dar, după experienţa mea, apelul iniţial la funcţia `mysql_pconnect()`are cele mai mari şanse să eşueze; ca atare, exemplele din acest capitol elimină numai mesajele generate de funcţia respectivă.

Poate că nu vă convine că numele şi parola sunt înglobate în script, în văzul tuturor. Ar cam fi cazul... Este adevărat că numele şi parola nu apar în pagina Web care este transmisă clientului, deoarece conţinutul scriptului este înlocuit de datele sale de ieşire. Totuşi, dacă serverul Web suferă o eroare de configurare şi nu reuşeşte să recunoască faptul că scriptul dumneavoastră trebuie prelucrat de PHP, va trimite scriptul sub formă de text simplu, iar parametrii dumneavoastră de conexiune vor fi expuşi. Vom aborda în curând această problemă, în secţiunea „Utilizarea funcţiilor şi a fişierelor `include`".

Identificatorul de legătură returnat de funcţia `mysql_pconnect()` poate fi transferat către alte numeroase apeluri la funcţii conexe MySQL din interfaţa API pentru PHP. Totuşi, pentru asemenea, apeluri, identificatorul este întotdeauna opţional.

`mysql_pconnect()` sau `mysql_connect()`

O funcţie care este similară cu `mysql_pconnect()` este `mysql_connect()`. Ambele preiau ca argumente numele de gazdă, numele utilizatorului şi parola şi ambele returnează `FALSE` pentru a indica reuşita sau eşecul încercării de conexiune. Diferenţa dintre cele două apeluri este că `mysql_pconnect()` stabileşte o conexiune persistentă, în timp ce `mysql_connect()` stabileşte o conexiune non-persistentă. O conexiune persistentă diferă de o conexiune non-persistentă în sensul că prima nu este închisă la terminarea scriptului. Dacă un alt script PHP este executat ulterior de acelaşi proces Apache copil şi apelează funcţia `mysql_pconnect()` cu aceleaşi argumente, conexiunea este refolosită. Acest procedeu este cu mult mai eficient decât stabilirea fiecărei conexiuni pornind de la zero.

De exemplu, puteţi apela `mysql_select_db()` folosind oricare din următoarele forme:

```
mysql_select_db ($db_name, $link);
mysql_select_db ($db_name);
```

[2] În traducere *Avertisment: conexiune MySQL ratată: accesul interzis pentru utilizatorul ...* (*Foloseşte parolă:* DA). – N.T.

Dacă omiteţi argumentul `link` din orice apel PHP legat de MySQL care preia un asemenea argument, apelul foloseşte conexiunea deschisă cel mai recent. Astfel, dacă scriptul dumneavoastră deschide numai o singură conexiune, nu trebuie să specificaţi niciodată în mod explicit un argument `link` în nici unul din apelurile dumneavoastră MySQL – conexiunea respectivă va fi cea prestabilită. Situaţia este complet diferită de programarea în MySQL folosind interfeţele API în C sau DBI, pentru care nu există o asemenea valoare prestabilită.

Am scris programul de conexiune în scriptul nostru de creare a unei pagini de bază simple după cum se poate vedea mai jos, pentru a clarifica tipul de valoare returnat de funcţia `mysql_pconnect()`:

```
$link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
    or exit ();
```

Totuşi, nu folosim parametrul `$link` nicăieri în cadrul scriptului, deci programul ar fi putut fi scris mai simplu astfel:

```
@mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
    or exit ();
```

Presupunând că stabilirea conexiunii a reuşit, următoarea etapă este selectarea unei baze de date:

```
mysql_select_db ("samp_db")
    or exit ();
```

Dacă apelul la funcţia `mysql_select_db()` eşuează, scriptul îşi încheie execuţia în tăcere. O eroare este improbabilă în acest moment, dacă am reuşit să ne conectăm la server şi dacă baza de date există, dar este prudent să verificăm apariţia eventualelor probleme şi să luăm măsurile adecvate. După ce aţi selectat baza de date, putem trimite interogarea spre server, extragem rezultatul, îl afişăm şi eliberăm memoria alocată setului de rezultate:

```
$result = mysql_query ("SELECT COUNT(*) FROM membru")
    or exit ();
if ($row = mysql_fetch_array ($result))
    echo "<P>In prezent, Liga are " . $row[0] . " membri";
mysql_free_result ($result);
```

Funcţia `mysql_query()` trimite interogarea la server pentru a fi executată. Interogarea nu trebuie să se termine cu un caracter punct şi virgulă şi nici cu grupul de caractere `\g`. Funcţia `mysql_query()` returnează `FALSE` dacă interogarea a fost ilegală sau dacă nu a putut fi executată dintr-un motiv oarecare. În caz contrar, returnează un identificator al setului de rezultate. Acest identificator este o valoare pe care o putem folosi pentru a obţine informaţii despre setul de rezultate. Pentru interogarea noastră, setul de rezultate constă dintr-un singur rând cu o singură valoare din coloană, reprezentând numărul de membri. Pentru a obţine această valoare, transferăm identificatorul setului de rezultate funcţiei `mysql_fetch_array()` pentru a prelua rândul, atribuim rândul variabilei `$row` şi obţinem accesul la primul element (care se întâmplă să fie şi singurul) sub forma `$row[0]`.

Când am terminat cu setul de rezultate, eliberăm memoria alocată acestuia transferându-l funcţiei `mysql_free_result()`. De fapt, acest apel nu este necesar în scriptul nostru, deoarece PHP eliberează automat memoria alocată tuturor seturilor de rezultate active la terminarea execuţiei unui script.

Funcţia `mysql_free_result()` este utilă mai ales în scripturile care execută interogări foarte mari sau un număr mare de interogări şi previne utilizarea unei cantităţi excesive de memorie.

Pentru a folosi scriptul nostru, trebuie să-l instalăm undeva. În acest capitol, am adoptat convenţia prin care Liga istorică americană îşi are propriul său catalog, denumit `lisua`[3], situat la nivelul cel mai de sus al arborelui de documente Apache, deci scriptul paginii de bază poate fi instalat sub forma `lisua/index.php` în arborele respectiv. Vom crea scripturi şi pentru proiectul de evidenţă a rezultatelor şcolare, deci îi vom atribui acestuia catalogul `evid`. Dacă gazda sitului Web este `pit-viper.snake.net`, paginile din aceste două cataloage au adrese URL care încep astfel:

```
http://pit-viper.snake.net/lisua/
http://pit-viper.snake.net/evid/
```

De exemplu, paginile de bază din fiecare catalog pot fi denumite `index.php` şi pot fi accesibile astfel:

```
http://pit-viper.snake.net/lisua/index.php
http://pit-viper.snake.net/evid/index.php
```

Utilizarea funcţiilor şi a fişierelor `include`

Scripturile PHP diferă de scripturile DBI, prin aceea că scripturile PHP sunt localizate în arborele cu documente al serverului dumneavoastră de Web, în timp ce scripturile DBI se găsesc, în mod caracteristic, într-un catalog `cgi-bin` care se află în afara arborelui cu documente. Astfel, apare o problemă de securitate: o eroare de configurare a serverului poate determina afişarea accidentală pentru clienţi, sub formă de text simplu, a paginilor localizate în arborele cu documente. Aceasta înseamnă că numele de utilizator şi parolele pentru stabilirea conexiunilor cu serverul MySQL sunt într-un pericol mai mare de a fi expuse lumii exterioare dacă sunt folosite într-un script PHP decât dacă se folosesc într-un script DBI.

Variabile în PHP

În PHP puteţi declara variabile prin simpla utilizare a acestora. Scriptul nostru de creare a unei pagini de bază foloseşte trei variabile, şi anume `$link`, `$result` şi `$row`, din care nici una nu este declarată nicăieri. (Există contexte în care trebuie să declaraţi variabilele, cum ar fi atunci când faceţi referire la o variabilă globală în interiorul unei funcţii, dar despre aceasta vom discuta ulterior.)

Variabilele sunt simbolizate printr-un identificator precedat de simbolul dolarului ($). Această notaţie este valabilă indiferent de valoarea pe care o reprezintă variabila, deşi pentru tablouri şi obiecte trebuie să mai adăugaţi şi alte elemente pentru a obţine accesul la elementele individuale ale unei valori. Dacă o variabilă `$x` reprezintă o singură valoare, cum ar fi un număr sau un şir, se poate obţine acces la aceasta sub forma `$x`. Dacă `$x` reprezintă un tablou cu indici numerici, elementele sale sunt accesibile sub forma `$x[0]`, `$x[1]` etc. Dacă `$x` reprezintă un tablou cu indici asociativi precum `"galben"` sau `"mare"`, elementele sale sunt accesibile sub forma `$x["galben"]` sau `$x["mare"]`.

Tablourile PHP pot avea atât elemente numerice, cât şi elemente asociative. De exemplu, atât `$x[1]` cât şi `$x["mare"]` pot reprezenta elemente ale aceluiaşi tablou. Dacă `$x` reprezintă un obiect, are proprietăţi care sunt accesibile sub forma `$x->nume_proprietate`. De exemplu, `$x->galben` şi `$x->mare` pot fi proprietăţi ale variabilei `$x`. Numerele nu se pot folosi ca nume de proprietăţi, deci `$x->1` nu este o construcţie corectă în PHP.

[3] Abreviere personală pentru Liga Istorică a Statelor Unite ale Americii. – N.T.

Scriptul nostru inițial pentru construcția paginii de bază a Ligii istorice este expus la această problemă, deoarece conține valorile literale ale numelui de utilizator și ale parolei MySQL. Să mutăm acești parametri de conexiune în afara scriptului, folosind două dintre caracteristicile limbajului PHP: funcții și fișiere `include`. Vom scrie o funcție `samp_db_connect()` pentru a stabili conexiunea, respectiv vom plasa funcția într-un fișier *include* – un fișier care nu face parte din scriptul nostru principal, dar la care se poate face referire din acesta. Unele avantaje ale aceste abordări sunt:

- **Programul de stabilire a conexiunii este mai ușor de scris.** Nu avem nevoie să scriem toți parametrii și putem cere funcției `samp_db_connect()` să selecteze baza de date automat după conectare, astfel încât o funcție să execute atribuțiile a două funcții PHP. Astfel, scripturile devin mai ușor de înțeles, deoarece atenția dumneavoastră se poate concentra asupra aspectelor caracteristice ale scriptului, fără a fi deturnată de programul de configurare a conexiunii.
- **Fișierul `include` este accesibil din scriptul nostru, dar poate fi deplasat în afara arborelui cu documente Apache.** Ca atare, conținutul său devine inaccesibil clienților și aceștia nu mai pot avea acces la parametrii de conexiune, chiar dacă serverul Web suferă o eroare de configurare. Utilizarea unui fișier `include` este o bună strategie pentru ascunderea oricăror categorii de informații importante, care nu doriți să fie trimise în afara sitului de către serverul dumneavoastră de Web.

Aceasta nu înseamnă, totuși, că numele și parola sunt sigure din toate punctele de vedere. Utilizatorii care pot deschide o sesiune de lucru la gazda serverului Web pot fi capabili să citească fișierul `include` în mod direct, dacă nu luați nici o măsură de precauție. Consultați secțiunea „Conectarea la serverul MySQL din scripturile Web" din capitolul 7, „Interfața API pentru Perl DBI" și rețineți măsurile de precauție descrise acolo pentru instalarea fișierelor de configurare DBI astfel încât acestea să fie protejate de alți utilizatori. Aceleași măsuri de precauție se aplică și în cazul utilizării fișierelor `include` din PHP.

Influențe lingvistice în PHP

Dacă aveți experiență cu programarea în C, probabil ați observat că numeroase construcții sintactice din scriptul nostru sunt foarte similare celor pe care le folosiți pentru programarea în C. De fapt, sintaxa PHP derivă în mare măsură din C, deci asemănarea nu este o coincidență. Dacă aveți oarecare cunoștințe de C, le veți putea transfera în mare măsură în PHP. De fapt, dacă nu sunteți sigur cu privire la modul de scriere a unei expresii sau structuri de control în PHP, încercați să o scrieți în C și este posibil să fie corectă.

Deși PHP derivă cu precădere din C, sunt prezente și elemente de Java și Perl. Puteți constata acest lucru din sintaxa comentariilor, care permite oricare din următoarele trei forme:

```
# Comentariu in stil Perl,de la # pana la sfarsitul liniei
// Comentariu in stil C++ sau Java, de la // pana la sfarsitul liniei
/* Comentariu in stil C, intre slash-stea pana la stea-slash */
```

Alte asemănări cu Perl includ operatorul `.` de concatenare a șirurilor (inclusiv `.=` drept operator de concatenare aditivă), precum și modalitatea în care referințele la variabile și secvențele escape sunt interpretate în interiorul ghilimelelor duble, dar nu și în interiorul ghilimelelor simple.

- **Fișierul `include` poate fi folosit în mai multe scripturi.** Astfel este promovată reutilizarea programelor, iar liniile de program devin mai ușor de întreținut. De asemenea, permite efectuarea cu ușurință de modificări globale în fiecare script care obține acces la fișier. De exemplu, dacă mutăm baza de date `samp_db` de la `pit-viper` la `boa`, nu trebuie să modificăm o mulțime de scripturi individuale, ci numai argumentul „numele gazdei" din apelul la funcția `mysql_pconnect()` din fișierul `include` care conține funcția `samp_db()`.

Pentru a folosi fișiere `include`, trebuie să aveți unde să le puneți și trebuie să cereți limbajului PHP să le caute. Dacă sistemul dumneavoastră dispune deja de o asemenea locație, o puteți folosi. Dacă nu, folosiți următoarea procedură pentru a stabili o locație a fișierului `include`:

1. Creați un catalog în care veți stoca fișierele PHP `include`. Acest catalog nu trebuie să se afle în interiorul arborelui cu documente al serverului Web! Eu folosesc un catalog `include` pentru PHP `/usr/local/apache/php`, care se află la același nivel ca și arborele meu cu documente (`/usr/local/apache/htdocs`), dar nu în interiorul acestuia.
2. Creați referințe la fișierele `include` cu numele complet al căii de acces sau indicați limbajului PHP cataloagele în care să le caute. Ultima metodă este mai convenabilă, deoarece, dacă folosim numele de bază al fișierului, PHP îl va găsi automat.[4] Pentru a indica limbajului PHP unde să caute, modificați fișierul de inițializare PHP (`/usr/local/lib/php3.ini` în sistemul meu) pentru a schimba valoarea parametrului `include_path`. Dacă acesta nu are nici o valoare, atribuiți-i calea de acces completă a noului catalog `include`:

   ```
   include_path = "/usr/local/apache/php"
   ```

   Dacă `include_path` are deja o valoare, adăugați noul catalog la valoarea respectivă:

   ```
   include_path = "valoare_curenta:/usr/local/apache/php"
   ```

3. Creați fișierul `include` pe care doriți să-l folosiți și inserați-l în catalogul `include`. Fișierul trebuie să aibă un nume distinctiv; pentru noi, numele `samp_db.inc` este suficient. Conținutul fișierului este prezentat în listingul următor. Pentru scripturile pe care le scriem aici, când ne conectăm la serverul MySQL, vom folosi întotdeauna baza de date `samp_db`, deci funcția de conexiune `samp_db_connect()` poate selecta automat acea bază de date. Funcția returnează un identificator de legătură dacă reușește să se conecteze și dacă selectează baza de date, respectiv `FALSE` dacă se produce vreo eroare. Funcția nu afișează nici un mesaj dacă se produce o eroare, ceea ce permite apelantului să-și încheie discret execuția sau să afișeze un mesaj, în funcție de circumstanțe:

```
<?php
# samp_db.inc
# functiile comune ale bazei de date demonstrative samp_db

# se conecteaza cu serverul MySQL folosind numele si parola
# noastra ultrasecrete
function samp_db_connect ()
```

Continuare

[4] Fișierele `include` din PHP sunt oarecum asemănătoare cu fișierele antet din C, inclusiv sub aspectul că PHP le va căuta în numeroase cataloage, exact așa cum pre-procesorul C caută fișierele antet C în mai multe cataloage. – N.A.

Continuare

```
{
   $link = @mysql_pconnect("pit-viper.snake.net", "paul", "secret");
   if ($link && mysql_select_db ("samp_db"))
      return ($link);
   return (FALSE);
}
?>
```

Observați că fișierul `samp_db.inc` este delimitat prin etichetele `<?php` și `?>`. Aceasta deoarece PHP începe citirea fișierelor `include` în modul HTML. Fără etichete, PHP va trimite fișierul ca text simplu, fără a-l interpreta drept program PHP. Dacă intenționați să includeți cod HTML literal în fișier, nici o problemă. Dar, dacă fișierul conține linii de program PHP, trebuie să delimitați codul folosind etichetele de script.

4. Pentru a face referire la un fișier dintr-un script, folosiți o linie ca aceasta:

```
include ("samp_db.inc");
```

Când PHP vede linia respectivă, caută fișierul și îi citește conținutul. Toate componentele fișierului devin accesibile pentru componentele următoare ale scriptului.

După ce am configurat fișierul nostru `include samp_db.inc`, putem modifica pagina de bază a Ligii istorice astfel încât să facă referire la fișierul `include` și să se conecteze la serverul MySQL apelând funcția `samp_db_connect()`:

```
<HTML>
<HEAD>
<TITLE>Liga istorica americana</TITLE>
</HEAD>
<BODY>
<P>Bine ati venit in situl Web al Ligii istorice.
<?php
include ("samp_db.inc");

samp_db_connect ()
   or exit ();
$result = mysql_query ("SELECT COUNT(*) FROM membru")
   or exit ();
if ($row = mysql_fetch_array ($result))
```

include() sau require()

PHP are o funcționalitate `require()` care este similară cu `include()`. Deosebirea între ele este că, în cazul funcției `include()`, fișierul este citit și evaluat la fiecare execuție a instrucțiunii `include()`. În cazul funcției `require()`, fișierul este prelucrat o singură dată (de fapt, conținutul fișierului înlocuiește instrucțiunea `require()`). Aceasta înseamnă că, dacă doriți să citiți un alt fișier de fiecare dată când vă executați programul sau aveți un ciclu care parcurge un set de fișiere, aveți nevoie de `include()`, deoarece puteți configura o variabilă cu numele fișierului pe care doriți să-l includeți și să folosiți variabila ca argument al funcției `include()`.


```
   echo "<P>In prezent, Liga are " . $row[0] . " membri";
mysql_free_result ($result);
?>
</BODY></HTML>
```

Poate credeți că nu am economisit un număr chiar atât de mare de linii de program în pagina de bază folosind un fișier `include`. Așteptați puțin... Fișierul `samp_db.inc` va fi util și pentru alte funcții și îl putem folosi ca un depozit convenabil pentru diverse alte lucruri. De fapt, mai putem crea chiar acum alte două funcții care vor fi inserate în fișierul respectiv. Fiecare script pe care îl scriem va genera un set oarecum stereotip de etichete HTML la începutul paginii, respectiv un alt set la sfârșitul acesteia. În loc de a le scrie literal în fiecare script, putem scrie funcții `html_begin()` și `html_end()` care scriu etichetele automat. Funcția `html_begin()` poate prelua două argumente, care specifică titlul paginii și antetul acesteia. Codul pentru cele două funcții se prezintă astfel:

```
# Scrie etichetele HTML initiale pentru pagina. Se presupune
# ca eventualele caractere speciale din variabilele $titlu si
# $antet, daca exista, sunt codificate in mod corespunzator.

function html_begin ($titlu, $antet)
{
   print ("<HTML>\n");
   print ("<HEAD>\n");
   if ($titlu)
      print ("<TITLE>$titlu</TITLE>\n");
   print ("</HEAD>\n");
   print ("<BODY>\n");
   if ($antet)
      print ("<H2>$antet</H2>\n");
}

# scrie etichetele HTML finale ale paginii.

function html_end ()
{
   print ("</BODY></HTML>\n");
}
```

Apoi, putem modifica pagina de bază a Ligii istorice astfel încât să folosească cele două noi funcții și să aibă ca atare următorul aspect:

```
<?php
include ("samp_db.inc");

$titlu = "Liga istorica americana";
html_begin ($titlu, $titlu);
?>
```

Continuare

Continuare

```
<P>Bine ati venit in situl Web al Ligii istorice.

<?php
samp_db_connect ()
  or exit ();
$result = mysql_query ("SELECT COUNT(*) FROM membru")
  or exit ();
if ($row = mysql_fetch_array ($result))
  echo "<P>In prezent, Liga are " . $row[0] . " membri";
mysql_free_result ($result);

html_end ();
?>
```

Observați că programul a fost împărțit în două componente, iar textul HTML literal al mesajului de întâmpinare este inserat între cele două părți.

Utilizarea funcțiilor pentru generarea componentelor inițială și finală ale paginii ne oferă o posibilitate importantă. Dacă dorim să modificăm aspectul antetului sau al subsolului tuturor paginilor care folosesc aceste funcții, putem include pur și simplu în funcții unele linii de program. Fiecare script care va folosi funcțiile va fi automat afectat. De exemplu, puteți dori să scrieți un mesaj „Drepturile de autor rezervate LISUA" în partea de jos a fiecărei pagini din situl Web al Ligii istorice. O funcție de final de pagină, cum este `html_end()`, constituie o modalitate simplă de a face această schimbare.

O pagină cu o interogare simplă

Scriptul pe care l-am înglobat în pagina de bază a sitului Ligii istorice rulează o interogare care returnează un singur rând. Următorul nostru script prezintă modul de prelucrare a unui set de rezultate compus din mai multe linii. Scriptul preia și afișează conținutul tabelului `membru`. Acesta este echivalentul PHP al scriptului DBI `dump_members` pe care l-am creat în capitolul 7, deci îl vom denumi `dump_members.php`. Acest script este diferit de versiunea DBI prin aceea că este destinat utilizării într-un mediu Web, nu de la linia de comandă. Din acest motiv, trebuie să producă date de ieșire în format HTML, nu doar să scrie un simplu text delimitat prin tabulatori. Pentru a alinia în mod estetic liniile și coloanele, vom scrie înregistrările membrilor într-un tabel HTML. Scriptul se prezintă astfel:

```
<?php
# dump_members.php - afiseaza lista membrilor Ligii istorice

include ("samp_db.inc");

$titlu = "Lista membrilor Ligii istorice americane";
html_begin ($titlu, $titlu);

samp_db_connect ()
  or die ("Nu se poate conecta la server");
```



```
# emite interogarea
$query = "SELECT nume, prenume, sufix, email,"
   . "strada, oras, stat, cod_postal, telefon"
   . "FROM membru ORDER BY nume";
$result = mysql_query ($query)
  or die ("Nu poate executa interogarea");

print ("<TABLE>\n");
# citeste rezultatele interogarii, apoi face curatenie
while ($row = mysql_fetch_row ($result))
{
  print ("<TR>\n");
  for ($i = 0; $i < mysql_num_fields ($result); $i++)
  {
    # modifica semnificatia caracterelor speciale si afiseaza
    printf ("<TD>%s</TD>\n", htmlspecialchars ($row[i]));
  }
  print ("</TR>\n");
}
mysql_free_result ($result);
print ("</TABLE>\n");

html_end ();
?>
```

Acest script folosește funcția `die()` pentru a afișa un mesaj și pentru a termina execuția scriptului dacă se produce vreo eroare.[5] Aceasta este o altă modalitate de tratare a erorilor decât aceea folosită în pagina de bază a Ligii istorice, unde execuția scriptului s-a încheiat în tăcere. În scriptul `dump_members.php`, ne așteptăm să vedem un anumit rezultat, deci este logic să afișăm un mesaj de eroare prin care se indică apariția unei probleme.

Scriptul poate fi instalat în catalogul `lisua` și este accesibil sub forma `http://pit-viper.snake.net/lisua/dump_members.php`. Apoi, putem adăuga o legătură la noul nostru script în pagina de bază a Ligii istorice, astfel încât publicul să fie la curent cu acesta:

```
<?php
include ("samp_db.inc");

$titlu = "Liga istorica americana";
html_begin ($titlu, $titlu);
?>

<P>Bine ati venit la situl Web al Ligii istorice.
```

Continuare

[5] Funcția `die()` este similară cu funcția `exit()`, dar afișează un mesaj înainte de a-și încheia execuția. – N.A.

Continuare

```
<?php
samp_db_connect ()
  or exit ();
$result = mysql_query ("SELECT COUNT(*) FROM membru")
  or exit ();
if ($row = mysql_fetch_array ($result))
  echo "<P>In prezent, Liga are " . $row[0] . " membri";
mysql_free_result ($result);
?>

<P>
Puteti consulta catalogul cu membri
<A HREF="dump_members.php">aici</A>.

<?php
html_end ();
?>
```

Prelucrarea rezultatelor interogării

În această secţiune, vom examina mai detaliat modul de executare a interogărilor MySQL şi manipularea seturilor de rezultate. În PHP, toate interogările sunt emise prin apelarea funcţiei `mysql_query()`, care preia ca argumente un şir de interogare şi un identificator de legătură. Identificatorul de legătură este opţional, deci puteţi invoca funcţia `mysql_query()` folosind oricare dintre următoarele forme:

```
$result = mysql_query ($query, link);  # foloseste conexiunea
                                       # explicita
$result = mysql_query ($query);        #foloseste conexiunea
                                       # prestabilita
```

Pentru interogările care nu returnează rânduri (interogări de tip non-`SELECT`, precum `DELETE`, `INSERT`, `REPLACE` şi `UPDATE`), funcţia `mysql_query()` returnează `TRUE` sau `FALSE` pentru a indica reuşita sau eşecul interogării. Pentru o interogare reuşită, puteţi apela funcţia `mysql_affected_rows()` pentru a determina câte rânduri au fost modificate (adică şterse, inserate, înlocuite sau actualizate, după caz).

Pentru instrucţiunile `SELECT`, `mysql_query()` returnează un identificator al setului de rezultate sau `FALSE` pentru a indica reuşita, respectiv eşecul interogării. Pentru o interogare reuşită, folosiţi identificatorul setului de rezultate pentru a obţine alte informaţii despre setul de rezultate. De exemplu, puteţi afla numărul de rânduri şi de coloane din setul de rezultate, respectiv puteţi prelua rândurile incluse în set.

Când `mysql_query()` returnează `FALSE` (adică zero), înseamnă că o interogare a eşuat – cu alte cuvinte, s-a produs o eroare şi interogarea nu a putut fi nici măcar executată. O interogare poate eşua din mai multe motive:

- Poate conţine malformaţii sau o eroare de sintaxă.
- Interogarea poate fi corectă sub aspect sintactic, dar poate fi lipsită de sens din punct de vedere semantic, cum ar fi încercarea de a selecta dintr-un tabel o coloană pe care acesta nu o conţine.
- Nu aveţi suficiente privilegii pentru a efectua interogarea.
- Gazda serverului MySQL a devenit inaccesibilă, datorită unor probleme în reţea.

În fiecare din aceste cazuri (mai sunt şi altele), funcţia `mysql_query()` returnează `FALSE`. Dacă doriţi să cunoaşteţi motivul erorii, apelaţi una din funcţiile `mysql_error()` sau `mysql_errno()` pentru a obţine şirul mesajului de eroare sau codul numeric al erorii. (Vezi „Tratarea erorilor".)

Două din cele mai comune greşeli care se comit în utilizarea funcţiei `mysql_query()` sunt aceea de a crede că valoarea returnată este un număr de rânduri, respectiv că valoarea returnată conţine datele returnate de interogarea dumneavoastră. Nici una din aceste afirmaţii nu este adevărată.

Tratarea interogărilor care nu returnează nici un set de rezultate

Programul următor foloseşte funcţia `DELETE` pentru a ilustra modul de prelucrare al unei interogări care nu returnează nici un rând:

```
$result = mysql_query ("DELETE FROM membru WHERE membru_id = 149");
if (!$result)
  print ("interogarea a esuat\n");
else
  printf("numar de randuri sterse: %d\n", mysql_affected_rows ());
```

Dacă există un membru al cărui identificator este 149, MySQL şterge înregistrarea şi funcţia `mysql_query()` returnează `TRUE`. Ce se întâmplă, însă, dacă un asemenea membru nu există? În acest caz, `mysql_query()` returnează tot `TRUE`! Acest fapt îi va surprinde pe aceia care persistă în greşeala de a crede că valoarea returnată de `mysql_query()` este un număr de rânduri. Nu este. Valoarea returnată este `TRUE` în ambele situaţii deoarece interogarea este corectă, indiferent dacă şterge sau nu vreun rând. Numărul de rânduri afectat de interogare este cu totul şi cu totul altceva. Pentru a obţine valoarea respectivă după o interogare reuşită, apelaţi funcţia `mysql_affected_rows()`.

Tratarea interogărilor care returnează un set de rezultate

Exemplul următor furnizează o schiţă generală a modului de prelucrare a interogărilor `SELECT`:

```
$result = mysql_query ("SELECT * FROM membru");
if (!$result)
  print ("interogarea a esuat\n");
else
{
  printf ("numarul de randuri returnate: %d\n",
    mysql_num_rows ($result));
```

Continuare

Continuare

```
    while ($row = mysql_fetch_row ($result))
    {
        for ($i = 0; $i < mysql_num_fields ($result); $i++)
        {
            if ($i > 0)
                print (",");
            print ($row[$i]);
        }
        print ("\n");
    }
    mysql_free_result ($result);
}
```

Nu presupuneți că `mysql_query()` va reuși

În lista de corespondență referitoare la PHP, utilizatorii începători de PHP întreabă de ce apare următorul mesaj de eroare la execuția unui script:

```
Warning: 0 is not a MySQL result index in file on line n
```

Acest mesaj arată că o valoare egală cu zero a identificatorului setului de rezultate a fost transferată unei funcții oarecare (cum ar fi o funcție de preluare a rândurilor) care presupune un set de rezultate valid. Aceasta înseamnă că un apel anterior la `mysql_query()` a returnat zero, adică `FALSE`. Cu alte cuvinte, interogarea `mysql_query()` a eșuat și autorul scriptului nu s-a deranjat să verifice valoarea returnată înainte de a o folosi la alte funcții. Când folosiți `mysql_query()`, testați întotdeauna valoarea returnată.

Dacă interogarea eșuează, rezultatul este `FALSE` și afișăm un mesaj în acest sens. (Este posibil să existe și alte răspunsuri mai adecvate la apariția unei erori, în funcție de circumstanțe.) Dacă interogarea reușește, funcția `mysql_query()` returnează un identificator al setului de rezultate. Această valoare returnată este utilă în anumite moduri (dar *nu* ca număr de rânduri!). Identificatorul setului de rezultate poate fi folosit în oricare din următoarele scopuri:

- Se transferă funcției `mysql_num_rows()` pentru a determina numărul de rânduri din setul de rezultate.
- Se transferă funcției `mysql_num_fields()` pentru a determina numărul de coloane din setul de rezultate.
- Se transferă unei rutine de preluare a rândurilor, pentru a prelua rânduri succesive din setul de rezultate. Exemplul nostru folosește funcția `mysql_fetch_row()`, dar există și alte opțiuni, despre care vom discuta curând.
- Se transferă funcției `mysql_free_result()`, pentru a elibera memoria alocată setului de rezultate și pentru a permite limbajului PHP să elibereze toate resursele alocate acestuia.

PHP furnizează numeroase funcții de preluare a rândurilor pentru regăsirea unui set de rezultate după ce funcția `mysql_query()` a executat cu succes o interogare `SELECT` (vezi tabelul 8.1). Fiecare din aceste funcții preia un identificator al setului de rezultate ca argument și returnează `FALSE` atunci când nu mai există rânduri.

Tabelul 8.1 Funcții PHP de preluare a rândurilor

Numele funcției	Valoare returnată
`mysql_fetch_row()`	Un tablou; elementele sunt accesibile prin intermediul indicilor numerici
`mysql_fetch_array()`	Un tablou; elementele sunt accesibile prin intermediul indicilor numerici sau asociativi
`mysql_fetch_object()`	Un obiect; elementele sunt accesibile prin intermediul proprietăților

Cel mai elementar apel este cel la funcția `mysql_fetch_row()`, care returnează următorul rând al setului de rezultate sub forma unui tablou. Elementele tabloului sunt accesibile cu ajutorul indicilor numerici cu valori cuprinse între 0 și `mysql_num_fields() - 1`. Exemplul următor prezintă modul de utilizare a funcției `mysql_fetch_row()` într-un ciclu simplu, care preia și afișează valorile din fiecare rând:

```
$query = "SELECT * FROM presedinte";
$result = mysql_query ($query)
    or die ("Interogarea a esuat");
while ($row = mysql_fetch_row ($result))
{
    for ($i = 0; $i < mysql_num_fields ($result); $i++)
    {
        if ($i > 0)
            print ("\t");
        print ($row[$i]);
    }
    print ("\n");
}
```

Variabila `$row` este un tablou. Elementele din acest tablou sunt accesibile sub forma `$row[$i]`, unde `$i` este indexul numeric al coloanei. Dacă sunteți familiarizat cu funcția `count()` din PHP, puteți fi tentat să o folosiți pentru a determina numărul de elemente din tablou în loc de a folosi `mysql_num_fields()`. Nu o faceți. Funcția `count()` numără numai elementele din tablou care sunt configurate, iar PHP nu configurează elemente care corespund unor valori din coloane egale cu `NULL`. Funcția `count()` este o măsură inadecvată a numărului de coloane returnate, deoarece funcția nu are această finalitate. Considerația este valabilă și pentru celelalte două funcții de preluare a rândurilor.

Cea de-a doua funcție de preluare a rândurilor prezentată în tabelul 8.1, și anume `mysql_fetch_array()`, este similară cu `mysql_fetch_row()`, cu deosebirea că elementele din tabloul pe care îl returnează sunt accesibile atât cu ajutorul indexului numeric, cât și prin intermediul indexului asociativ. Cu alte cuvinte, puteți obține acces la elemente atât prin număr, cât și după nume:

```
$query = "SELECT nume, prenume FROM presedinte";
$result = mysql_query ($query)
    or die ("Interogarea a esuat");
while ($row = mysql_fetch_array ($result))
```

Continuare

Continuare

```
{
   printf ("%s %s\n"), $row[0], $row[1]);
   printf ("%s %s\n"), $row["prenume"], $row["nume"]);
}
```

Informațiile returnate de funcția `mysql_fetch_array()` reprezintă un superset al informațiilor returnate de `mysql_fetch_row()`. În ciuda acestui fapt, diferențele de performanță între cele două funcții sunt neglijabile și puteți apela la funcția `mysql_fetch_array()` fără nici o diminuare semnificativă a performanțelor.

Cea de-a treia funcție de preluare a rândurilor, `mysql_fetch_object()`, returnează următorul rând al setului de rezultate sub forma unui obiect. Aceasta înseamnă că obțineți accesul la elementele rândului folosind sintaxa `$row-nume_coloana`. De exemplu, dacă regăsiți valorile `nume` și `prenume` din tabelul `presedinte`, puteți obține accesul la aceste coloane folosind sintaxa `$row-nume` și `$row-prenume`:

```
$query = "SELECT nume, prenume FROM presedinte";
$result = mysql_query ($query)
   or die ("Interogarea a esuat");
while ($row = mysql_fetch_object ($result))
   printf ("%s %s\n"), $row->prenume, $row->nume);
```

Testarea existenței valorilor NULL în rezultatele interogării

Pentru a verifica dacă o valoare dintr-o coloană returnată de o interogare `SELECT` este sau nu `NULL`, folosiți funcția `isset()`. De exemplu, dacă rândul dumneavoastră este inclus în tabloul `$row`, funcția `isset($row[$i])` are valoarea `FALSE` dacă expresia `$row[$i]`corespunde unei valori `NULL`, respectiv `TRUE` dacă expresia este diferită de `NULL`. O funcție conexă este `empty()`, dar aceasta returnează același rezultat pentru `NULL` ca și pentru șirurile vide, deci nu este utilă ca test pentru detectarea unei valori `NULL`.

Dacă interogarea dumneavoastră conține coloane cu valori calculate? De exemplu, puteți emite o interogare care returnează valori calculate ca rezultat al unei expresii:

```
SELECT CONCAT(prenume, " ", nume) FROM presedinte
```

O interogare scrisă astfel nu poate fi utilizată cu funcția `mysql_fetch_object()`. Numele coloanei selectate este expresia însăși, care nu constituie un nume valabil de proprietate. Totuși, puteți furniza un nume valabil atribuind coloanei un alias. Interogarea următoare atribuie coloanei aliasul `nume_complet`, care permite accesul la coloană folosind sintaxa `$row-nume_complet`, dacă preluați rezultatele cu ajutorul funcției `mysql_fetch_object()`:

```
SELECT CONCAT(prenume, " ", nume) AS nume_complet FROM presedinte
```

Tratarea erorilor

Pentru tratarea erorilor, PHP vă pune la dispoziție trei mijloace:

- **Utilizarea caracterului `@` pentru a elimina mesajele de eroare.** Puteți folosi caracterul `@` cu orice funcție care poate afișa un mesaj. Am procedat astfel când am apelat funcția `mysql_pconnect()` pentru a împiedica apariția mesajelor de eroare generate de funcția respectivă în pagina trimisă clientului.

- **Utilizarea funcției `error_reporting()`.** Această funcție activează sau dezactivează caracteristica de raportare a erorilor la oricare din următoarele nivele:

Nivel de eroare	Tipuri de erori raportate
1	Erori normale de funcții
2	Avertismente normale
4	Erori ale analizorului
8	Înștiințări

Pentru a controla caracteristica de raportare a erorilor, apelați funcția `error_reporting()` cu un argument egal cu suma nivelelor pe care doriți să le activați. Dezactivarea avertismentelor de nivel 1 și 2 este suficientă pentru a elimina mesajele provenite de la funcțiile MySQL:

```
error_reporting(4 + 8);
```

Probabil că nu doriți să dezactivați avertismentele de nivel 4 cu privire la erorile de analiză; dacă o faceți, veți avea mari probleme la depanarea oricăror modificări pe care le efectuați în scripturile dumneavoastră! Deseori, avertismentele de nivel 8 pot fi ignorate, dar uneori indică o problemă în scriptul dumneavoastră la care trebuie să fiți atent, deci este de dorit ca și acest nivel să rămână activat. Există și nivelele de eroare 16 și 32, dar acestea provin de la mecanismul de bază al limbajului PHP și nu de la funcții, deci în mod normal nu trebuie să vă preocupe.

- **Utilizarea funcțiilor `mysql_error()` și `mysql_errno()`.** Aceste funcții raportează informațiile de eroare care sunt returnate de serverul MySQL. Funcțiile respective sunt similare cu funcțiile din interfața API pentru C. `mysql_error()` returnează un mesaj de eroare sub formă de șir (un șir vid dacă nu s-a produs nici o eroare). `mysql_errno()` returnează un număr de eroare (0 dacă nu s-a produs nici o eroare). Ambele funcții preiau ca argument un identificator de legătură, care specifică o conexiune cu serverul MySQL, și returnează informații de eroare pentru funcția MySQL cel mai recent invocată din acea conexiune care returnează o stare. Identificatorul de legătură este opțional; dacă lipsește, se va folosi conexiunea cel mai recent deschisă. De exemplu, puteți raporta o eroare din `mysql_query()` în următorul mod:

```
if (!($result = mysql_query ( ... )))
{
   print ("errno: " . mysql_errno());
   print ("error: " . mysql_error());
}
```

Versiunile PHP ale funcțiilor `mysql_error()` și `mysql_errno()` diferă de omoloagele lor din C dintr-un important punct de vedere. În C, puteți obține informații despre eroare chiar și când o încercare de conectare la server eșuează. Prin contrast, apelurile PHP nu returnează informații utile pentru o conexiune decât după stabilirea cu succes a conexiunii. Cu alte cuvinte, dacă o încercare de conexiune eșuează, nu puteți folosi `mysql_errno()` sau `mysql_error()` pentru a afla motivul. Dacă doriți ca pentru eșecul conexiunii să raportați un anumit motiv, nu un mesaj generic, trebuie să luați măsuri speciale. Consultați Anexa H, „Referință API PHP", pentru instrucțiuni privind modul în care să procedați.

Scripturile din acest capitol afișează mesaje de eroare cu un conținut destul de general, precum „interogarea a eșuat", atunci când detectează vreo eroare. Totuși, când dezvoltați un script, deseori veți găsi utilă adăugarea unui apel la funcția `mysql_error()`, pentru a afla motivul specific al unei erori.

Probleme legate de ghilimele

Când construiți șiruri de interogări în PHP, este necesar să cunoașteți problemele legate de ghilimele, ca și în C sau Perl. Modalitatea în care sunt rezolvate problemele privind ghilimelele este, de asemenea, similară, deși numele funcțiilor sunt diferite în limbaje diferite.

Să presupunem că dumneavoastră construiți o interogare pentru inserția unei noi înregistrări într-un tabel; puteți delimita prin ghilimele o valoare care urmează a fi inserată într-o coloană șir:

```
$nume = "O'Malley";
$prenume = "Brian";
$data_expirare = "2002-9-1";
$query = "INSERT membru (nume,prenume,data_expirare)"
    . " VALUES('$nume','$prenume','$data_expirare')"
```

În acest caz, problema este că una dintre valorile încadrate între ghilimele conține alte ghilimele (`"O'Malley"`), ceea ce are ca rezultat o eroare de sintaxă dacă trimiteți interogarea la serverul MySQL. Pentru a rezolva problema în C, am fi apelat funcția `mysql_escape_string()`. Într-un script Perl DBI, am fi folosit `quote()`. PHP are o funcție denumită `addslashes()`, care execută cam aceeași operație. De exemplu, un apel la funcția `addslashes("O'Malley")` returnează valoarea `"O\'Malley"`. Exemplul anterior trebuie scris după cum urmează, pentru a se preveni problemele legate de ghilimele:

```
$nume = addslashes ("O'Malley");
$prenume = addslashes ("Brian");
$data_expirare = addslashes ("2002-9-1");
$query = "INSERT membru (nume,prenume,data_expirare)"
    . " VALUES('$nume','$prenume','$data_expirare')"
```

Metoda DBI `quote()` adaugă ghilimele de delimitare la șirul de interogare. Funcția `addslashes()` nu procedează astfel, deci trebuie să precizăm ghilimelele în mod explicit în șirul de interogare, în jurul valorilor care vor fi inserate.

Probleme privind ghilimelele se produc, de asemenea, când se scriu informațiile care vor fi prezentate în paginile Web. Dacă scrieți un șir care trebuie să apară sub formă de HTML sau ca parte a unui URL, cel mai bine este să îl codificați, dacă șirul poate conține caractere care sunt speciale pentru HTML sau în adrese URL. Funcțiile PHP `htmlspecialchars()` și `urlencode()` pot fi folosite în acest scop. Cele două funcții sunt asemănătoare cu metodele `escapeHTML()`, respectiv `escape()` din modulul CGI.pm.

Utilizarea PHP

În continuarea acestui capitol, vom aborda scopurile pe care ni le-am stabilit în capitolul 1 și pe care le mai avem de îndeplinit:

- Pentru proiectul de evidență a rezultatelor școlare, trebuie să scriem un script care să ne permită să introducem și să edităm punctajele obținute la teste și chestionare.
- Pentru Liga istorică, dorim să creăm un chestionar în format electronic despre președinții Statelor Unite și să-i imprimăm un caracter interactiv, astfel încât întrebările să poată fi generate imediat pentru vizitatorii sitului Web.
- De asemenea, dorim să permitem membrilor Ligii istorice să-și editeze rubricile aferente din catalogul în versiunea electronică, pentru a păstra informațiile actualizate și pentru a reduce volumul editărilor pe care le executăm personal.

Fiecare dintre aceste scripturi generează mai multe pagini Web și comunică de la o invocare a scriptului la alta prin intermediul informațiilor înglobate în paginile pe care le creează. Dacă nu cunoașteți conceptul de comunicații inter-pagini, puteți citi secțiunea „Scrierea paginilor cu destinație multiplă" din capitolul 7.

Introducerea punctajelor elevilor

În această secțiune, ne vom îndrepta atenția spre proiectul de evidență a rezultatelor școlare. Adresa URL a acestei zone în situl nostru de Web este `http://pit-viper.snake.net/evid/` și probabil că vom scrie pentru aceasta o scurtă pagină de bază `index.php`. Deocamdată, pagina următoare este suficientă. Pagina conține o legătură spre scriptul `score_browser` pe care l-am scris în capitolul 7, deoarece acel script are legătură cu proiectul de evidență a rezultatelor școlare:

```
<?php
include ("samp_db.inc");

$titlu = "Proiect de evidenta a rezultatelor scolare";
html_begin ($titlu, $titlu);
?>

<P>
<A HREF="/cgi-bin/score_browse">Vizualizarea</A> punctajelor
  obtinute la chestionare si teste

<?php
html_end ()
?>
```

Acum, să luăm în considerare proiectarea și implementarea scriptului `score_entry.php`, care ne va permite să introducem un set nou de punctaje obținute la chestionare sau teste, respectiv să edităm un set de înregistrări existent. Ultima funcționalitate este necesară pentru manipularea punctajelor elevilor care susțin un test sau un chestionar mai târziu decât restul clasei, datorită unei absențe pe motiv de boală sau dintr-un alt motiv (sau – Doamne ferește! – în caz că am introdus incorect punctajul). Schița scriptului pentru introducerea punctajelor este următoarea:

1. Pagina inițială prezintă o listă cu evenimente de examinare cunoscute și vă permite să alegeți unul, respectiv să specificați necesitatea de creare a unui eveniment nou.

2. Dacă optați pentru crearea unui eveniment nou, scriptul prezintă o pagină care vă permite să specificați data și tipul evenimentului. După crearea înregistrării aferente evenimentului, scriptul afișează din nou pagina cu lista de evenimente, pentru a prezenta noul eveniment.
3. Când alegeți un eveniment, scriptul prezintă o pagină de introducere a punctajului, care conține informațiile despre eveniment în partea de sus (identificator de eveniment, dată, tip), urmat de o listă cu o intrare pentru fiecare elev. Pentru evenimentele noi, intrările vor fi vide. Pentru evenimentele existente, intrările vor prezenta punctajul fiecărui elev. Când selectați butonul „Trimite", punctajele vor fi introduse în tabelul `puncte`.

Scriptul trebuie să execute numeroase operații diferite, ceea ce înseamnă că va trebui să transferăm o variabilă de stare de la o pagină la alta, pentru ca scriptul să știe ce are de făcut atunci când este invocat. În PHP, această operație este simplă, deoarece PHP prelucrează variabilele transferate ca parametri URL și le convertește în variabile cu același nume ca și parametrii. De exemplu, putem codifica un parametru `actiune` situat la sfârșitul adresei URL a scriptului nostru astfel:

```
http://pit-viper.snake.net/evid/score_entry.php?actiune=valoare
```

Când scriptul `score_entry.php` este invocat în acest mod, parametrul `actiune` este codificat sub forma variabilei `$actiune`, care este accesibilă direct. Acest fapt este valabil și pentru câmpurile din formulare. Să presupunem că un formular conține câmpuri denumite `nume` și `adresa`. Când un client trimite formularul, serverul Web invocă un script pentru a prelucra conținutul formularului. Scriptul poate afla valorile introduse în formular prin verificarea valorilor variabilelor `$nume` și `$adresa`. Pentru formulare care conțin un număr mare de câmpuri, nu este convenabil ca toate să aibă nume unice. PHP simplifică transferul tablourilor înăuntrul și în afara formularelor. Dacă folosiți nume de câmpuri precum `x[0]`, `x[1]` și așa mai departe, PHP le va codifica sub formă de elemente ale tabloului `$x`. Aceste element sunt accesibile sub forma `$x[0]`, `$x[1]` etc.

Vom comunica informațiile de la o invocare a scriptului `score_entry.php` la următoarea folosind în paginile noastre un parametru denumit `$actiune`, iar valoarea sa va fi verificată folosind variabila `$actiune` din scriptul nostru. Cadrul scriptului se prezintă astfel:

```
<?php
# score_entry.php - script de introducere a punctajelor pentru
# proiectul de evidenta a rezultatelor scolare

include ("samp_db.inc");

# defineste constantele de actiune
define (INITIAL_PAGE, 0);
define (SOLICIT_EVENT, 1);
define (ADD_EVENT, 2);
define (DISPLAY_SCORES, 3);
define (ENTER_SCORES, 4);
/* ... functiile se insereaza aici ... */
```



```
$titlu = "Proiect de evidenta a rezultatelor scolare -
          introducere punctaje";
html_begin ($titlu, $titlu);

samp_db_connect()
  or die ("Nu se poate conecta la serverul de baze de date")

if (empty ($actiune))
  $actiune = INITIAL_PAGE;

switch ($actiune)                 # ce avem de facut?
{
case INITIAL_PAGE:                # prezinta pagina initiala
  display_events ();
  break;
case SOLICIT_EVENT:               # solicita informatii privind noul
                                  # eveniment
  solicit_event_info ();
  break;
case ADD_EVENT:                   # adauga un eveniment nou la baza de date
  add_new_event ();
  display_events ();
  break;
case DISPLAY_SCORES:              # afiseaza punctajele pentru
                                  # evenimentul selectat
  display_scores ();
  break ();
case ENTER_SCORES: ()             # introduce punctajele noi
                                  # sau pe cele editate
  enter_scores ();
  display_events ();
  break;
default:
  die ("Cod de actiune ($actiune) necunoscut");
}

html_end ()
?>
```

Variabila `$actiune` poate lua numeroase valori, pe care le testăm în instrucțiunea `switch()`. (Pentru a evita să folosim numere literale în script, vom folosi construcția PHP `define()` pentru definirea constantelor.) Instrucțiunea `switch()` din PHP este similară cu omoloaga sa din C. În scriptul `score_entry.php`, această instrucțiune este folosită pentru a determina acțiunea care trebuie întreprinsă și pentru apelarea funcțiilor care implementează acțiunea.

Să examinăm pe rând toate funcțiile care execută aceste acțiuni. Prima, `display_events()`, regăsește rânduri din tabelul `eveniment` din MySQL și le afișează. Fiecare rând al tabelului afișează identificatorul de eveniment, data și tipul evenimentului (test sau chestionar), unde identificatorul de eveniment este scris sub forma unei legături pe care o puteți selecta pentru a edita punctajele obținute la evenimentul respectiv:

```
function display_events ()
{
global $PHP_SELF;

    print ("Selectati un eveniment executand clic pe numarul sau,\n");
    print ("sau selectati Eveniment nou pentru a crea un nou
            eveniment de examinare:<BR><BR>\n");
    $query = "SELECT eveniment_id, data, tip"
            . "FROM eveniment ORDER BY eveniment_id";
    $result = mysql_query ($query)
        or die ("Nu poate executa interogarea");
    print ("<TABLE BORDER>\n");
    print ("<TR>\n");
    display_cell ("TH", "Identificator de eveniment", 1);
    display_cell ("TH", "Data", 1);
    display_cell ("TH", "Tip", 1);
    print ("</TR>\n");

    # asociaza fiecare eveniment cu o legatura care va prezenta
    # punctajele aferente evenimentului; foloseste mysql_fetch_array ()
    # pentru a putea face referire la coloane in functie de numele acestora.
    while ($row = mysql_fetch_array ($result))
    {
        print ("<TR>\n");
        $url = sprintf ("%s?actiune=%d&eveniment_id=%d",
            $PHP_SELF, DISPLAY_SCORES, $row["eveniment_id"]);
        display_cell ("TD",
            "<A HREF=\"$url\">" . $row["eveniment_id"] . "</A>",
            0);
        display_cell ("TD", $row["data"], 1);
        display_cell ("TD", $row["tip"], 1);
        print ("</TR>\n");
    }
    # mai adauga o legatura pentru "Eveniment nou"
    print ("<TR ALIGN=CENTER>\n");
    $url = sprintf ("%s?actiune=%d", $PHP_SELF, SOLICIT_EVENT);
    display_cell ("TD COLSPAN=3",
                "<A HREF=\"$url\">" . "Eveniment nou" . "</A>",
                0);
    print ("</TR>\n");
```



```
    print ("</TABLE>\n");
}
```

Legăturile din tabel sunt construite folosind `$PHP_SELF`. Această variabilă conține adresa URL pentru script, ceea ce oferă scriptului o modalitate convenabilă de a se auto-invoca din nou. Cu toate acestea, rețineți linia `global` plasată aproape de începutul funcției:

```
global $PHP_SELF;
```

În funcțiile PHP, variabilele globale nu sunt accesibile decât dacă declarați în mod explicit că doriți să le folosiți. Fără linia `global`, `$PHP_SELF` ar fi fost tratată ca o variabilă locală (care este vidă, deoarece nu-i atribuim nici o valoare). De asemenea, este necesară utilizarea variabilei `global` în interiorul unei funcții, pentru a face referire la parametrii transferați scriptului dumneavoastră prin intermediul parametrilor URL sau sub formă de câmpuri din formular.

Funcția `display_cell()` folosită pentru generarea tabelului este similară cu funcția DBI cu același nume pe care am scris-o în capitolul 7[6]. Versiunea PHP se prezintă astfel:

```
function display_cell ($eticheta, $valoare, $codare)
{
    if ($codare)
        $valoare = htmlspecialchars ($valoare);
    if ($valoare == "")
        $valoare = " ";
    print ("<$eticheta>$valoare</$eticheta>\n");
}
```

Dacă selectați legătura „Eveniment nou" din tabelul pe care îl prezintă funcția `display_event()`, scriptul este invocat din nou, cu acțiunea `SOLICIT_EVENT`. Această acțiune determină un apel la funcția `solicit_event_info()`, care afișează un formular ce vă permite să introduceți informații pentru noul eveniment:

```
function solicit_event_info ()
{
global $PHP_SELF;
    printf ("<FORM METHOD=\"post\" ACTION=\"%s?actiune=%d\">\n",
            $PHP_SELF, ADD_EVENT);
    print ("Introduceti informatii pentru noul
            eveniment de examinare:<BR><BR>\n");
    print ("Data: <INPUT TYPE=text NAME=\"data\"");
    print (" VALUE=\"\" SIZE=10> ";)
    print ("Tip: ");
    print ("<INPUT TYPE=\"radio\" NAME=\"tip\" VALUE=\"T\" CHECKED>Test\n");
    print ("<INPUT TYPE=\"radio\" NAME=\"tip\" VALUE=\"C\">Chestionar\n");
    print ("<BR><BR>\n");
    print ("<INPUT TYPE=\"Trimite\" NAME=\"buton\" VALUE=\"Trimite\">\n");
    print ("</FORM>\n");
}
```

[6] Cu deosebirea că în capitolul 7 am folosit echivalentul românesc al numelui funcției – N.T.

Formularul generat de funcția `solicit_event_info()` conține un câmp de editare pentru introducerea datelor, o pereche de butoane radio pentru a specifica dacă intrarea este un test sau un chestionar, precum și un buton „Trimite". Când trimiteți formularul, este invocat scriptul `score_entry.php` cu acțiunea `ADD_EVENT`. Funcția `add_new_event()` este apelată pentru introducerea unui nou rând în tabelul `eveniment`:

```
function add_new_event()
{
global $data, $tip;

    if (empty ($data)) # verifica introducerea unei date
        die ("Nu a fost specificata nici o data");
    $query = sprintf ("INSERT INTO eveniment (data,tip)
            VALUES(\"%s\",\"%s\"), addslashes ($data), addslashes ($tip);
    if (!mysql_query ($query))
        die ("Nu a putut adauga evenimentul");
}
```

În funcția `add_new_event()`, folosim cuvântul cheie `global` pentru a obține accesul la câmpurile care sunt utilizate în formularul de introducere a evenimentelor noi (`data` și `tip`, accesibile sub forma variabilelor `$data` și `$tip`). După o verificare minimală de siguranță, pentru a ne asigura că datele nu sunt vide, introducem o nouă înregistrare în tabelul `eveniment`. Programul principal afișează din nou lista de evenimente după ce am introdus înregistrarea de eveniment, astfel încât să puteți selecta noul eveniment și să puteți începe introducerea punctajelor.

Funcția `display_scores()` se ocupă de căutarea tuturor punctajelor existente pentru un eveniment dat și afișează un formular care afișează punctajele, alături de numele elevilor:

```
function display_scores()
{
    # selecteaza punctele pentru evenimentul dat
    $query = "
        SELECT
            elev.elev_id, elev.nume, eveniment.data,
            puncte.puncte AS puncte, eveniment.tip
        FROM elev, eveniment
            LEFT JOIN puncte ON elev.elev_id = puncte.elev_id
                AND eveniment.eveniment_id = puncte.eveniment_id
        WHERE eveniment.eveniment_id = $eveniment_id
        ORDER BY elev.nume
    ";
    $result = mysql_query ($query)
        or die ("Nu poate executa interogarea");

    printf ("<FORM METHOD=\"post\" ACTION=\"%s?actiune=%d&eveniment_id=%d\">\n",
        $PHP_SELF, ENTER_SCORES, $eveniment_id);
```



```
    # afiseaza punctajele intr-un tabel si afiseaza data si tipul
    # evenimentului anterior tabelului. Totusi, nu putem afisa data si
    # tipul decat dupa preluarea primului rand din setul de rezultate

    $needheading = 1;
    while ($row = mysql_fetch_array ($result))
    {
        if ($needheading)
        {
            printf ("Identificator eveniment: %s,
                    Data eveniment: %s, Tip eveniment: %s\n,
                    $eveniment_id, $row["data"], $row["tip"]);
            print ("<BR><BR>\n");
            print ("<TABLE BORDER>\n");
            print ("<TR>\n");
            display_cell ("TH", "Nume", 1);
            display_cell ("TH", "Puncte", 1);
            print ("</TR>\n");
            $needheading = 0;
        }
        print ("<TR>\n");
        display_cell ("TD", $row["nume"], 1);
        $col_val = sprintf ("<INPUT TYPE=text NAME=\"puncte[%s]\"",
            $row["elev_id"]);
        $col_val .= sprintf (" VALUE=\"%s\" SIZE=5><BR>\n",
            $row["puncte"]);
        display_cell ("TD", $col_val, 0);
        print ("</TR>\n");
    }

    print ("</TABLE>\n");
    print ("<BR>\n");
    print ("<INPUT TYPE=\"Trimite\" NAME=\"buton\" VALUE=\"Trimite\">\n");
    print ("</FORM>\n");
}
```

Interogarea pe care funcția `display_scores()` o folosește pentru regăsirea informațiilor privind punctajele nu este o simplă unire între tabele, deoarece aceasta nu va selecta un rând al unui elev care nu a obținut nici un punctaj la evenimentul respectiv. În particular, pentru un eveniment nou, unirea respectivă nu va selecta nici o înregistrare și vom avea un formular de introducere vid! Dorim să folosim un `LEFT JOIN` pentru a forța regăsirea unui rând pentru fiecare elev, indiferent dacă elevul are sau nu un punctaj în tabelul `puncte`. Elementele fundamentale ale unei interogări similare celei pe care funcția `display_scores()` o folosește pentru regăsirea înregistrărilor cu punctaje din MySQL au fost date în capitolul 3, „Sintaxa și utilizarea SQL în MySQL", în secțiunea

„Verificarea valorilor inexistente într-un tabel". Interogarea de acolo selecta numai punctaje lipsă; aici vom selecta numai punctajele de la un anumit eveniment.

Punctajele sunt incluse în formular folosind câmpuri cu nume precum `puncte[n]`, unde `n` este o valoare `elev_id`. Când formularul este trimis înapoi la serverul Web, PHP va converti aceste câmpuri în elemente ale tabloului `$puncte` și vom putea obține acces la elementele tabloului pentru a reface conținutul formularului.

Când terminați introducerea sau editarea punctajelor și trimiteți formularul, scriptul `score_entry.php` este invocat cu acțiunea `ENTER_SCORES` și este apelată funcția `enter_scores()` pentru a prelucra informațiile din formular:

```
function enter_scores ()
{
global $puncte, $eveniment_id;

    $incorect = 0;
    $blank = 0;
    $nonblank = 0;
    while (list ($elev_id, $punctajnou) = each ($score))
    {
        $punctajnou = trim ($punctajnou);
        if (empty ($punctajnou))   # fara punctaj,
                                   # sterge daca exista in tabel
        {
            ++$blank;
            $query = "DELETE FROM puncte"
                . " WHERE eveniment_id = $eveniment_id"
                . " AND elev_id = $elev_id";
        }
        else if (ereg ("^[0-9]+$", $punctajnou))
            # trebuie sa fie intreg
        {
            ++$nonblank;
            $query = "REPLACE INTO puncte (eveniment_id,"
                . " elev_id,puncte) VALUES(eveniment_id,"
                . " $elev_id,$punctajnou)";
        }
        else
        {
            ++$incorect;
            continue;
        }
        if (!mysql_query ($query))
            die ("insertie punctaje ratata, identificator de eveniment
                $eveniment_id, identificator de elev $elev_id)";
    }
    printf ("Punctaje introduse: %d<BR>\n", $nonblank);
    printf ("Punctaje lipsa: %d<BR>\n", $blank);
    printf ("Punctaje incorecte: %d<BR>\n", $incorect);
    printf ("<BR>\n");
}
```


Valorile identificatorului de elev și punctajele asociate sunt obținute prin parcurgerea ciclică a tabloului `puncte` folosind funcția PHP `each`. Fiecare punctaj este prelucrat după cum urmează:

- Dacă punctajul este vid, nu este nimic de introdus, dar încercăm să ștergem punctajul în cazul în care există. (Poate că anterior am introdus din greșeală un punctaj pentru un elev care de fapt lipsea.)
- Dacă punctajul nu lipsește, se execută o validare rudimentară a valorii. După eliminarea spațiilor albe inițiale și finale folosind funcția `trim()`, acceptăm rezultatul, dacă rămâne un blanc sau un număr întreg. Totuși, valorile din formulare sunt întotdeauna codificate sub formă de șiruri, astfel încât nu putem verifica dacă o valoare este sau nu de tip întreg folosind `is_long()` sau `is_int()`. Aceste funcții vor returna `FALSE` chiar dacă valoarea conține numai cifre. În acest caz, o corespondență cu un model funcționează mai bine. Testul următor are valoarea `TRUE` dacă fiecare caracter, de la începutul la sfârșitul șirului, este o cifră:

```
(ereg ("^[0-9]+$", $str)
```

  Dacă punctajul este OK, îl adăugăm la tabelul `punctaj`. Interogarea folosește `REPLACE` în loc de `INSERT`, deoarece s-ar putea să înlocuim un punctaj existent, în loc de a introduce unul nou. (Instrucțiunea `REPLACE` se poate folosi în ambele cazuri.)

Astfel, am rezolvat problema scriptului `score_entry.php`. Toate operațiile de introducere a punctajelor și de editare se pot face acum din browserul dumneavoastră de Web. Un dezavantaj evident este acela că scriptul nu asigură nici o măsură de securitate; orice persoană care se poate conecta la serverul Web are posibilitatea de a edita punctajele. Scriptul pe care îl vom scrie ulterior pentru editarea intrărilor aferente membrilor Ligii istorice prezintă o schemă de autentificare simplă care poate fi adaptată pentru acest script. De asemenea, puteți folosi pachetul PHPLIB pentru a furniza un mecanism de autentificare mai sofisticat.

Chestionar despre președinții americani

Unul dintre scopurile sitului Web al Ligii istorice a fost utilizarea acestuia pentru a prezenta o versiune electronică a unui chestionar, similară cu unele dintre chestionarele pe care Liga le publică în secțiunea pentru copii a buletinului său de știri, intitulat „Cronici ale trecutului SUA". De fapt, am creat tabelul `presedinte` pentru a-l putea folosi ca sursă de întrebări pentru un chestionar pe teme istorice. Pentru a prezenta acest chestionar, vom scrie un script denumit `pres_ches.php`.

Ideea de bază constă în a alege un președinte în mod aleator, a pune o întrebare despre el și apoi de a solicita un răspuns de la utilizator, respectiv de a vedea dacă răspunsul este corect sau nu. Pentru a nu ne complica, vom reduce subiectul la o întrebare privind locul

nașterii unui președinte. O altă operație simplificatoare este de a prezenta întrebările într-un format cu răspunsuri multiple. Astfel, treaba devine mai simplă pentru utilizator, care trebuie numai să selecteze dintr-un set de întrebări, în loc de a tasta un răspuns. De asemenea, este mai ușor și pentru noi, deoarece nu trebuie să facem comparații între șiruri pentru a determina ce anume a tastat utilizatorul. Avem nevoie de o comparație simplă între opțiunea utilizatorului și valoarea pe care o căutăm.

Un script care prezintă acest chestionar trebuie să îndeplinească două funcții. Mai întâi, pentru invocarea inițială, trebuie să genereze și să afișeze o nouă întrebare prin căutarea de informații în tabelul `presedinte`. În al doilea rând, dacă scriptul a fost apelat deoarece utilizatorul a trimis un răspuns, răspunsul trebuie verificat și este necesar un feedback pentru a semnala corectitudinea sau incorectitudinea răspunsului. În acest caz, scriptul trebuie să genereze și să afișeze o nouă întrebare. Dacă răspunsul a fost incorect, trebuie afișată din nou aceeași întrebare.

Pentru a genera întrebările, vom folosi una din caracteristicile care au apărut în MySQL 3.23, și anume `ORDER BY RAND()`. Folosind această funcție, putem selecta în mod aleator rânduri din tabelul `presedinte`. De exemplu, pentru a alege numele unui președinte și locul nașterii acestuia în mod aleator, interogarea următoare execută chiar operația de care aveți nevoie:

```
SELECT CONCAT(prenume, ' ', nume) AS nume,
   CONCAT(oras, ', ', stat) AS loc
   FROM presedinte ORDER BY RAND() LIMIT 1
```

Numele va fi acela al președintelui selectat, iar locul nașterii va fi răspunsul corect la întrebarea „Unde s-a născut acest președinte?". De asemenea, va trebui să prezentăm unele opțiuni incorecte, pe care le putem selecta folosind o interogare similară:

```
SELECT DISTINCT CONCAT(oras, ', ', stat) AS loc
   FROM presedinte ORDER BY RAND()
```

Din rezultatul acestei interogări vom selecta primele patru valori care diferă de răspunsul corect. Funcția care emite aceste interogări și regăsește rezultatele se prezintă astfel:

```
function config_ches ()
{
   # emite o interogare pentru a selecta un presedinte si pentru a
   # obtine locul nasterii acestuia
   $query = "SELECT CONCAT(prenume, ', ', nume) AS nume,"
      . " CONCAT(oras, ' ', stat) AS loc"
      . " FROM presedinte ORDER BY RAND() LIMIT 1";
   $result = mysql_query ($query)
      or die ("Nu poate executa interogarea");
   $row = mysql_fetch_array ($result)
      or die ("Nu poate prelua rezultatul");
   $nume = $row["nume"];
   $nume = $row["loc"];

   # Construieste setul de optiuni privind locul nasterii care vor fi
   # prezentate. Configureaza tabloul $alegere, care contine cinci locuri
   # de nastere, din care unul este raspunsul corect.
   $query = "SELECT DISTINCT CONCAT(oras, ',  , stat) AS loc"
         . " FROM presedinte ORDER BY RAND()";
   $result = mysql_query ($query)
      or die ("Nu poate executa interogarea");
   $alegere[] = $loc;              # initializeaza tabloul cu optiunea
                                   # corecta
   while (count ($alegere) < 5 && $row = mysql_fetch_array ($result))
   {
      if ($row["loc"] == $loc)
            continue;
      $alegere[] = $row["loc"]; # adauga o alta optiune
   }
   # asaza optiunile intr-o ordine aleatoare, apoi afiseaza formularul
   shuffle ($alegere);
   display_form ($nume, $loc, $alegere);
}
```


Pentru a prezenta informațiile privind întrebările din chestionar, vom folosi un formular care afișează numele președintelui, un set de butoane radio care conțin opțiunile posibile și un buton „Trimite". Acest formular trebuie să execute două operații: să prezinte clientului informațiile despre chestionar și să transmită acele informații înapoi serverului Web, atunci când utilizatorul trimite un răspuns, astfel încât să putem verifica dacă răspunsul a fost corect sau nu.

Pentru a determina formularul să efectueze aceste operații, vom include în formular informațiile privind chestionarul folosind numeroase câmpuri ascunse. Vom denumi câmpurile `nume`, `loc` și `alegere` pentru a reprezenta numele și locul nașterii unui președinte, respectiv setul de opțiuni posibile. Opțiunile pot fi codificate cu ușurință sub forma unui singur șir folosind funcția `implode()` pentru a concatena valorile la un loc, folosind un caracter special. (Avem nevoie de caracterul special astfel încât să putem defalca ulterior șirul folosind funcția `explode()` dacă trebuie să afișăm din nou întrebarea.) Funcția de afișare a formularului se prezintă astfel:

```
function display_form ($nume, $loc, $alegere)
{
global $PHP_SELF;

   printf ("<FORM METHOD=\"post\" ACTION=\"%s">\n", $PHP_SELF);
   hidden_field ("nume", $nume);
   hidden_field ("loc", $loc);
   hidden_field ("alegere", implode ("#", $alegere));
   printf ("Unde s-a nascut %s?<BR><BR>\n", $alegere);
   for ($i = 0; $i < 5; $i++)
   {
      print ("<INPUT TYPE=\"radio\" NAME=\"raspuns\"\n");
```

Continuare

Continuare

```
        printf ("VALUE=\"%s\">%s<BR>\n", $alegere[$i], $alegere[$i]);
    }
    print ("<BR><INPUT TYPE=\"Trimite\" VALUE=\"Trimite\">\n");
    print ("</FORM>\n");
}
```

Funcția `hidden_field()` scrie HTML pentru un câmp ascuns în forma:

```
function hidden_field ($nume, $valoare)
{
    printf ("<INPUT TYPE=\"HIDDEN\" NAME=\"%s\" VALUE=\"%s\">\n",
            $nume, $valoare);
}
```

Când utilizatorul face o anumită opțiune și trimite formularul, răspunsul este codificat sub forma valorii câmpului `raspuns` în cadrul conținutului formularului, conținut trimis înapoi la serverul Web. Putem descoperi valorile câmpurilor `nume`, `loc`, `alegere` și `raspuns` consultând valorile variabilelor `$nume`, `$loc`, `$alegere` și `$raspuns`. Astfel, avem și posibilitatea de a ne da seama dacă scriptul a fost apelat pentru prima dată sau dacă utilizatorul trimite un răspuns la un formular afișat anterior: nici una dintre aceste variabile nu va fi configurată dacă scriptul este acum invocat pentru prima dată. Astfel, corpul principal al scriptului determină ce are de făcut examinând una din aceste variabile:

```
if (!$nume)         # prima apelare
    config_ches();
else                # utilizatorul a trimis raspuns la formular
    verifica_raspuns ();
```

Mai trebuie să scriem funcția `verifica_raspuns()`, care compară răspunsul utilizatorului cu răspunsul corect. Am codificat răspunsul corect în câmpul `loc` al formularului, iar răspunsul utilizatorului se află în câmpul `raspuns`, deci tot ceea ce avem de făcut este să comparăm valorile variabilelor `$loc` și `$raspuns`. În funcție de rezultatul comparației, furnizăm un feedback oarecare și fie generăm și afișăm o nouă întrebare, fie afișăm din nou aceeași întrebare:

```
function verifica_raspuns ()
{
global $nume, $loc, $alegere;
global $raspuns;

    if ($loc == $raspuns)       # raspuns corect;
                                # genereaza o noua intrebare
    {
        print ("Raspunsul este corect!<BR>\n");
        printf ("%s s-a nascut in %s.<BR>\n", $nume, $loc);
        print ("Incercati intrebarea urmatoare:<BR><BR>\n");
        config_ches();
    }
```



```
    else                        # raspuns incorect; reafiseaza intrebarea
    {
        print ("Ati gresit. Va rugam incercati din nou.<BR><BR>\n");
        $alegere = explode ("#", $alegere);
        display_form ($nume, $loc, $alegere);
    }
}
```

Am terminat. Adăugați o legătură la scriptul `pres_ches.php` în pagina de bază a Ligii istorice, iar vizitatorii pot încerca chestionarul pentru o verificare a cunoștințelor.

Editarea electronică a intrărilor aferente membrilor Ligii istorice

Scriptul nostru final, `edit_member.php`, va permite membrilor Ligii istorice să-și editeze propriile lor intrări on-line. Apoi, membrii pot corecta sau actualiza informațiile privind statutul lor de membru ori de câte ori doresc, fără a fi obligați să contacteze secretariatul Ligii pentru a trimite modificările. Existența acestei posibilități ar trebui să contribuie la păstrarea într-o formă mai actualizată a catalogului membrilor și (nu este o coincidență!) la reducerea volumului de muncă a secretarului Ligii.

O măsură de precauție pe care trebuie să ne-o luăm este de a preveni modificarea intrărilor de către orice altă persoană în afara membrului căruia îi aparține intrarea. Aceasta înseamnă că avem nevoie de o formă oarecare de securitate. Pentru a ilustra o formă simplă de autentificare, vom folosi MySQL pentru a stoca parole pentru fiecare membru și pentru a cere membrilor să furnizeze o parolă corectă pentru a obține accesul la formularul de editare pe care îl prezintă scriptul nostru. Scriptul funcționează după cum urmează:

- Când este invocat pentru prima dată, `edit_member.php` prezintă un formular care conține câmpuri pentru identificatorul de membru și o parolă.
- Când este trimis formularul inițial, scriptul caută în tabela cu parole, folosind identificatorul de membru drept cheie pentru a găsi parola asociată. Dacă parola este găsită, scriptul caută intrarea aferentă membrului din tabelul `membru` și o afișează pentru editare.
- Când formularul editat este trimis, actualizăm intrarea folosind conținutul formularului.

Cadrul scriptului `edit_member.php` este următorul:

```
<?php
# edit_member.php - editeaza intrarile membrilor
# Ligii istorice prin intermediul Web-ului

include ("samp_db.inc");

# defineste constantele de actiune
define (INITIAL_PAGE, 0);
define (DISPLAY_ENTRY, 1);
define (UPDATE_ENTRY, 2);
```

Continuare

Continuare

```
/* ... puneti functiile aici ... */

if (empty ($actiune))
   $actiune = INITIAL_PAGE;

$titlu = "Liga istorica americana - formular de editare membri";
html_begin ($titlu, $titlu);

samp_db_connect ()
   or die ("Nu se poate conecta la server");

switch ($actiune)
{
case INITIAL_PAGE:
   solicit_member_id ();
   break;
case DISPLAY_ENTRY:
   display_entry ();
   break;
case UPDATE_ENTRY:
   update_entry ();
   break ();
default:
   die ("Cod de actiune necunoscut ($actiune)");
}

html_end ();
?>
```

Pagina inițială prezintă un formular în care se cere identificatorul membrului și parola:

```
function solicit_member_id ()
{
global $PHP_SELF;

   printf ("<FORM METHOD=\"post\" ACTION=\"%s?actiune=%d\">\n",
                                  $PHP_SELF, DISPLAY_ENTRY);
   print ("Introduceti numarul dv. de identificare si parola, \n");
   print ("apoi selectati Trimite.\n<BR><BR>\n");
   print ("<TABLE>\n");
   print ("<TR>");
   print ("<TD>Identificator de membru</TD><TD>");
   print ("<INPUT TYPE=text NAME=\"identificator_membru\" SIZE=10><BR>\n");
```



```
   print ("</TD></TR>");
   print ("<TR>");
   print ("<TD>Parola</TD><TD>");
   print ("<INPUT TYPE=password NAME=\"parola\" SIZE=10><BR>\n");
   print ("</TD></TR>");
   print ("</TABLE>\n");
   print ("<INPUT TYPE=\"Trimite\" NAME=\"buton\" VALUE=\"Trimite\">\n");
   print ("</FORM>\n");
}
```

Evident, avem nevoie de parole. O modalitate simplă de obținere a acestora este prin generare aleatoare. Următoarele instrucțiuni configurează un tabel `parole_membri`, apoi creează o parolă pentru fiecare membru prin generarea unei sume de control MD5 pornind de la un număr aleator. Probabil că veți permite membrilor să-și aleagă propriile parole, dar puteți invoca `mysql` și emiteți aceste instrucțiuni, ca o modalitate rapidă și simplă de a configura parolele:

```
CREATE TABLE parole_membri
(
   membru_id INT UNSIGNED NOT NULL PRIMARY KEY,
   parola CHAR(8)
)
INSERT INTO parole_membri (membru_id, parola)
   SELECT
      membru_id,
      LEFT(MD5(RAND()), 8) AS parola
   FROM membru
```

De asemenea, vom adăuga în acest tabel o intrare specială pentru membrul 0, cu o parolă care va servi drept parolă administrativă (superutilizator). Putem folosi această parolă pentru a obține accesul la orice intrare dorim:

```
INSERT INTO parole_membri (membru_id, parola) VALUES (0, "secret");
```

După ce ați creat tabelul cu parole, puteți dori să dezactivați scriptul `samp_browse` pe care l-ați scris în capitolul 7, care permite oricui să parcurgă conținutul oricărui tabel din baza de date `samp_db` – inclusiv al tabelului `parole_membri`.

Când un membru introduce un identificator și o parolă și trimite formularul, scriptul `edit_member.php` afișează intrarea în vederea editării:

```
function display_entry ()
{
global $PHP_SELF;
global $membru_id, $parola;

   $membru_id = trim ($membru_id);
   if (empty ($membru_id))
      die ("Nu a fost specificat nici un identificator de membru");
```

Continuare

Continuare

```
if (!ereg ("^[0-9]+$, $membru_id))          # trebuie sa fie intreg
    die ("S-a specificat un identificator de membru incorect
    (trebuie sa fie numar)");
if (empty ($parola))
    die ("Nu s-a specificat nici o parola");
if (check_pass ($membru_id, $parola))       # membru obisnuit
    $admin = 0;
else if (check_pass (0, $parola))           # administrator
    $admin = 1;
else
    die ("Parola incorecta");

$query = "SELECT nume, prenume, sufix, email,"
    . "strada, oras, stat, cod_postal, telefon, interese,"
    . "membru_id, data_expirare"
    . " FROM membru"
    . " WHERE membru_id = $membru_id"
    . " ORDER BY nume;
$result = mysql_query ($query)
    or die ("Nu poate executa interogarea");
if (mysql_num_rows ($result) == 0)
    die ("Nu a fost gasit nici un utilizator cu
    membru_id = $membru_id");
if (mysql_num_rows ($result) > 1)
    die ("S-au gasit mai multi utilizatori cu
    membru_id = $membru_id");
printf ("<FORM METHOD=\"post\" ACTION=\"%s?actiune=%d\">\n",
                $PHP_SELF, UPDATE_ENTRY);
hidden_field ("membru_id", $membru_id);
hidden_field ("parola", $parola);
print ("<TABLE>\n");
# citeste rezultatele interogarii si le formateaza pentru editare
$row = mysql_fetch_array ($result);
display_column ("Identificator de membru", $row, "membru_id", 0);
# permite utilizatorului cu parola administrativa
# sa modifice data expirarii
display_column ("Data expirarii", $row, "data_expirarii", $admin);
display_column ("Nume", $row, "nume", 1);
display_column ("Prenume", $row, "prenume", 1);
display_column ("Sufix", $row, "sufix", 1);
display_column ("Email", $row, "email", 1);
display_column ("Strada", $row, "strada", 1);
display_column ("Oras", $row, "oras", 1);
```



```
    display_column ("Stat", $row, "stat", 1);
    display_column ("Cod postal", $row, "cod_postal", 1);
    display_column ("Telefon", $row, "telefon", 1);
    display_column ("Interese", $row, "interese", 1);
    print ("</TABLE>\n");
    print ("<INPUT TYPE=\"Trimite\" NAME=\"buton\" VALUE=\"Trimite\">\n");
    print ("</FORM>\n");
}
```

Primul lucru pe care trebuie să-l facă funcţia `display_entry()` este să verifice parola. Dacă parola introdusă în formular este echivalentă cu parola din tabelul `parole_membri` pentru identificatorul de membru dat, respectiv dacă este similară cu parola administrativă (adică parola membrului 0), scriptul `edit_member.php` afişează intrarea pentru editare. Funcţia de verificare a parolelor `check_pass()` rulează o interogare simplă pentru a extrage o înregistrare din tabelul `parole_membri`:

```
function check_pass ($id, $parola)
{
    $query = "SELECT parola FROM parole_membri"
        . "WHERE membru_id = $id";
    if (!($result = mysql_query ($query)))
        die ("Eroare la citirea tabelului cu parole");
    if (!($row = mysql_fetch_array ($result)))
        return (FALSE);
    return ($row["parola"] == parola);  # TRUE daca parola corespunde
}
```

Formularul de editare afişează valoarea identificatorului de membru sub formă de text numai pentru citire, deoarece acesta nu trebuie modificat. Pentru membrii obişnuiţi, data expirării este de asemenea afişată sub formă de text numai pentru citire, deoarece nu dorim ca membrii să o modifice. Totuşi, dacă se furnizează parola administrativă, data de expirare devine editabilă, permiţând astfel secretarului Ligii să actualizeze data pentru membrii care şi-au plătit cotizaţiile.

Coloanele aferente intrării din tabelul `membru` sunt afişate de către funcţia `display_column()`. Funcţia adaugă coloane la formularul de editare sub formă de text editabil sau text numai pentru citire, în funcţie de valoarea celui de-al treilea argument al funcţiei:

```
# Afiseaza o coloana din intrarea unui membru. $eticheta este eticheta
# vizibila, afisata utilizatorului. $row este tabloul care contine
# intrarea. $nume_coloana este numele unei coloane din intrare. $valoare
# este valoarea din coloana. $editabil este diferit de zero daca
# utilizatorul are permisiunea de a modifica valoarea. Altfel, valoarea
# este afisata ca text non-editabil. Numele campurilor sunt construite
# cu sintaxa row[nume_coloana], astfel incat, atunci cand formularul
# este trimis, valorile pot fi accesibile prin intermediul unui tablou,
# nu prin intermediul unei multimi de variabile individuale.
```

Continuare

Continuare

```
function display_column ($eticheta, $row, $nume_coloana, $editabil)
{
   print ("<TR>\n");
   printf ("<TD>%s</TD>\n", htmlspecialchars ($eticheta));
   $valoare = htmlspecialchars ($row[$nume_coloana]);
   if ($editabil)  # afiseaza sub forma de camp de editare
   {
      $str = sprintf ("<INPUT TYPE=\"text\" NAME=\"row[%s]\"", $nume_coloana);
      $str .= sprintf (" VALUE=\"%s\" SIZE=\"80\">\n", $valoare);
   }
   else            # afiseaza sub forma de text numai pentru citire
      $str = $valoare;
   printf ("<TD>%s</TD>\n", $str);
   print ("</TR>\n");
}
```

Funcţia `display_entry()` înglobează valorile parametrilor `membru_id` şi `parola` sub formă de câmpuri ascunse în formular, astfel încât acestea să se păstreze pentru următoarea invocare a scriptului `edit_member.php`, când membrul îşi trimite intrarea modificată. Acest procedeu permite verificarea automată a parolei pentru un anumit identificator, fără a mai cere membrului să o introducă din nou. (Observaţi că metoda noastră simplă de autentificare implică transferul înainte şi înapoi al parolei în clar. În general, aceasta nu este o idee prea bună, dar Liga istorică nu impune măsuri ridicate de securitate, iar metoda respectivă este adecvată scopurilor noastre. Veţi avea nevoie de un mecanism mai puternic dacă executaţi tranzacţii financiare, de exemplu.)

Funcţia care actualizează intrarea se prezintă astfel:

```
function update_entry ()
{
global $row, $membru_id, $parola;

   $membru_id = trim ($membru_id);
   if (empty ($membru_id))
      die ("Nu a fost specificat nici un identificator de membru");
   if (!ereg ("^[0-9]+$, $membru_id)) # trebuie sa fie intreg
      die ("Identificator de membru incorect (trebuie sa fie numar)");
   if (!check_pass ($membru_id, $parola) && !check_pass (0, $parola))
      die ("Parola incorecta");

   # Vom avea nevoie de un set de rezultate pentru a evalua
   # posibilitatea ca in coloanele tabelului membru sa existe
   # valoarea NULL. Aceasta interogare ne ofera un asemenea
   # set fara a selecta nici un rand.
   $result = mysql_query ("SELECT * FROM membru WHERE 1 = 0");
   if (!$result)
      die ("Nu poate interoga tabelul membru");

   # parcurge ciclic fiecare camp din formular, folosind
   # valorile pentru a construi instructiunea UPDATE.

   $query = "UPDATE membru ";
   $delim = "SET ";           # pune "SET" inaintea primei coloane si ","
                              # inaintea celorlalte
   while (list ($col_name, $val) = each ($row))
   {
      $query .= "$delim $nume_coloana =";
      $delim = ",";

      # daca o valoare este vida, actualizeaza valoarea cu NULL, daca
      # este acceptata de coloana. Acest procedeu previne incercarea
      # de a pune un sir vid in coloana cu data expirarii, atunci cand
      # ar fi trebuit sa fie NULL, de exemplu.
      $val = trim ($val);
      if (empty ($val))
      {
            if (nullable ($result, $nume_coloana))
                  $query .= "NULL"; # introduce NULL
            else
                  $query .= "\"\""; # introduce sirul vid
      }
      else
            $query .= "\"" . addslashes ($val) . "\"";
   }
   $query .= " WHERE membru_id = $membru_id";
   if (mysql_query ($query) && mysql_affected_rows () > 0)
      print ("Actualizarea intrarii a reusit.\n");
}
```

Mai întâi, verificăm din nou parola, pentru a ne asigura că nimeni nu încearcă să ne păcălească prin trimiterea unui formular falsificat, după care intrarea este actualizată. Actualizarea necesită o oarecare atenţie, deoarece, dacă un câmp din formular este vid, poate fi necesar ca valoarea acestuia să fie introdusă ca `NULL`, nu sub formă de şir vid. Coloana `data_expirare` este un exemplu în acest sens. O dată de plată a cotizaţiei egală cu `NULL` are semnificaţia specială de „membru pe viaţă". Dacă inserăm un şir vid în această coloană, valoarea va fi convertită în `"0000-00-00"`, iar membrul respectiv îşi va pierde calitatea de membru pe viaţă.

Pentru a rezolva această problemă, căutăm metadatele coloanei şi verificăm dacă aceasta este declarată ca `NULL` sau `NOT NULL`. Aceste informaţii sunt returnate de funcţia `mysql_fetch_field()`; din păcate, funcţia respectivă caută coloanele în funcţie de indexul

numeric. Referirea la coloanele tabelului membru este mai convenabil, deci vom scrie o mică funcție intitulată nullable(), care preia un nume de coloană și găsește obiectul de metadate adecvat:

```
# Determina daca coloana cu numele dat din setul de rezultate
# accepta sau nu valoarea NULL.

function nullable ($result, $nume_coloana)
{
   for ($i = 0; $i < mysql_num_fields ($result); $i++)
   {
      if (!($fld = mysql_fetch_field ($result, $i)))
            continue;
      if ($fld->nume == $nume_coloana)
            return (!$fld->not_null);
   }
   return (0);
}
```

Funcția mysql_fetch_field() necesită un identificator al setului de rezultate pentru tabelul ale cărui coloane le verificăm. Acest identificator se poate obține executând o interogare SELECT normală, care nu returnează rânduri. Interogarea va returna un set de rezultate vid, dar acesta este un mijloc suficient de regăsire a metadatelor de care avem nevoie pentru estimarea capacității coloanelor din tabelul membru de a conține valori NULL:

```
SELECT * FROM membru WHERE 1 = 0
```

Asta-i tot. Instalați scriptul, comunicați-le membrilor parolele și aceștia își vor putea actualiza singuri informațiile aferente fiecăruia.

PARTEA A III-A

Administrarea sistemului MySQL

CAPITOLUL 9

Introducere în administrarea sistemului MySQL

Din ansamblul sistemelor de baze de date, MySQL este relativ simplu de utilizat, iar efortul necesar pentru instalarea și utilizarea sistemului MySQL este de asemenea redus. Simplitatea sistemului MySQL explică probabil o mare parte din popularitatea sa, mai ales în rândul celor care nu sunt și nici nu vor să fie programatori. MySQL ajută la formarea unor profesioniști competenți în domeniul calculatoarelor, desigur, dar aceasta nu este o cerință obligatorie pentru exploatarea unei instalări MySQL reușite.

Cu toate acestea, o instalare MySQL nu va rula de una singură, indiferent de nivelul dumneavoastră de experiență. Trebuie să existe cineva care să supravegheze sistemul, pentru a se asigura că funcționează uniform și eficient, precum și cineva care să știe cum să rezolve eventualele probleme. Dacă dumneavoastră sunteți cel (cea) care trebuie să vă asigurați că MySQL rulează în liniște acolo unde este instalat, citiți în continuare.

În Partea a III-a a acestei cărți, vom examina diferitele aspecte ale administrării sistemului MySQL. Capitolul de față furnizează o trecere în revistă a ceea ce trebuie să știți despre responsabilitățile pe care le implică administrarea unei instalări a sistemului MySQL. Capitolul curent furnizează o scurtă descriere a acestor responsabilități, iar capitolele următoare furnizează instrucțiuni pentru îndeplinirea lor.

Dacă sunteți un administrator MySQL începător sau lipsit de experiență, nu vă lăsați intimidat de lunga listă de responsabilități prezentate în acest capitol. Fiecare sarcină prezentată în secțiunea următoare este importantă, dar nu trebuie să le învățați pe toate dintr-o dată. Dacă preferați, puteți folosi capitolele din această parte a cărții ca referință, căutând subiectele atunci când descoperiți că trebuie să le cunoașteți.

Dacă aveți experiență în administrarea altor sisteme de baze de date, veți descoperi că rularea unei instalări a sistemului MySQL este similară din anumite puncte de vedere cu exploatarea acestor sisteme, precum și că experiența dumneavoastră este o resursă valoroasă. Dar administrarea sistemului MySQL își are propriile sale cerințe specifice; această parte a cărții vă va ajuta să vă familiarizați cu ele.

O trecere în revistă a atribuțiilor administrative

Sistemul de baze de date MySQL constă din numeroase componente. Trebuie să știți ce reprezintă aceste componente și care este rolul fiecăreia, fapt care vă va permite să înțelegeți atât natura sistemului pe care îl administrați, cât și instrumentele existente care sunt utile în activitatea dumneavoastră. Dacă vă acordați timp pentru a înțelege elementele sistemului MySQL pe care le controlați, munca dumneavoastră va fi mult

mai ușoară. În acest sens, trebuie să vă acomodați cu următoarele aspecte ale sistemului MySQL:

- **Serverul MySQL.** Serverul `mysqld` execută toate operațiile de manipulare a bazelor de date și a tabelelor. `safe_mysqld` este un program corelat, folosit pentru a porni serverul, a-l monitoriza și a-l reporni în caz de defecțiune.
- **Clienții și utilitarele MySQL.** Există numeroase programe MySQL care vă pot ajuta în comunicarea cu serverul și la îndeplinirea sarcinilor administrative. Cele mai importante dintre aceste programe sunt:
  - `mysql`, un program interactiv care vă permite să emiteți instrucțiuni SQL către server și să vizualizați rezultatele.
  - `mysqladmin`, un program administrativ care vă permite să îndepliniți sarcini precum închiderea serverului sau crearea, respectiv ștergerea bazelor de date. Mai puteți folosi `mysqladmin` pentru a verifica starea serverului, dacă vi se pare că acesta nu funcționează în mod adecvat.
  - `isamchk` și `myisamchk`, utilitare care vă ajută să efectuați analiza și optimizarea tabelelor, precum și refacerea în urma căderilor, dacă tabelele se deteriorează.
  - `mysqldump`, un instrument pentru executarea copiilor de siguranță a bazelor dumneavoastră de date sau pentru copierea acestora pe un alt server.
- **Limbajul serverului, SQL.** Unele îndatoriri administrative pot fi îndeplinite folosind numai utilitarul în linie de comandă `mysqladmin`, dar sunteți mai avantajat dacă puteți „discuta" cu serverul în propriul său limbaj. Ca un exemplu simplu, puteți fi pus în situația de a trebui să aflați de ce privilegiile unui utilizator nu funcționează în modul în care vă așteptați dumneavoastră. Nu există nici o alternativă față de comunicarea directă cu serverul. Puteți face aceasta folosind programul client `mysql`, pentru a emite interogări SQL care să vă permită examinarea tabelelor de acordare. Iar dacă versiunea dumneavoastră de MySQL este anterioară introducerii instrucțiunii `GRANT`, va trebui să folosiți `mysql` pentru a configura de la bun început privilegiile fiecărui utilizator.

  Dacă nu aveți cunoștințe de SQL, acumulați-le măcar pe cele fundamentale. O lipsă de fluență în SQL va avea ca efect numai o întârziere, în timp ce timpul necesar pentru învățare va fi compensat de mai multe ori. O cunoaștere autentică a limbajului SQL necesită oarecare timp, dar cunoștințele de bază pot fi acumulate rapid. Dacă aveți nevoie de unele îndrumări cu privire la SQL sau la folosirea clientului în linie de comandă `mysql`, consultați capitolul 1, „Introducere în MySQL și SQL".
- **Catalogul de date din MySQL.** Catalogul de date este locul în care serverul își stochează bazele sale de date și fișierele de stare. Este important să înțelegeți structura și conținutul catalogului de date, astfel încât să cunoașteți modul în care serverul folosește sistemul de fișiere pentru a reprezenta bazele de date și tabelele, precum și amplasarea unor fișiere precum jurnalele și conținutul acestora. De asemenea, trebuie să vă cunoașteți opțiunile pentru gestiunea alocării spațiului pe disc pe mai multe sisteme de fișiere, dacă descoperiți că sistemul de fișiere pe care este amplasat catalogul de date începe să devină prea aglomerat.

Noțiuni generale de administrare

Noțiunile generale de administrare se referă mai ales la operarea programului `mysqld`, serverul MySQL, precum și la furnizarea către utilizatorii dumneavoastră a accesului la server. Pentru îndeplinirea acestor responsabilități, următoarele îndatoriri sunt cele mai importante:

- **Pornirea și oprirea serverului.** Trebuie să puteți deschide și închide serverul manual, de la linia de comandă, precum și să configurați pornirea și oprirea automată a acestuia, atunci când sistemul dumneavoastră pornește, respectiv se oprește. De asemenea, este important de știut care sunt operațiile necesare pentru a readuce serverul în stare de funcționare după o cădere sau dacă nu pornește în mod adecvat.
- **Întreținerea conturilor de utilizator.** Trebuie să înțelegeți diferența dintre utilizatorii MySQL și utilizatorii Windows sau UNIX. Trebuie să știți să configurați conturile de utilizator MySQL, specificând utilizatorii care se pot conecta la server și sistemele de unde se pot conecta. Noii utilizatori trebuie să cunoască noii parametri de conexiune de care vor avea nevoie pentru a se conecta cu succes la server. Nu este treaba lor să determine modul în care le-ați configurat conturile!
- **Întreținerea fișierelor jurnal.** Trebuie să înțelegeți care sunt tipurile de fișiere jurnal pe care le puteți întreține, precum și momentul și modalitățile de întreținere a fișierelor jurnal. Rotația și expirarea fișierelor jurnal sunt esențiale pentru a împiedica fișierele jurnal să aglomereze sistemul dumneavoastră de fișiere.
- **Realizarea copiilor de siguranță și a copiilor simple ale bazelor de date.** Copiile de siguranță ale bazelor de date sunt de o importanță crucială în eventualitatea unei căderi grave a sistemului. Doriți să vă puteți restaura bazele de date în situația în care se aflau la momentul căderii, cu o pierdere minimă de date. Rețineți că executarea copiilor de siguranță a bazelor dumneavoastră de date nu este unul și același lucru cu executarea copiilor de siguranță generale ale sistemului, așa cum se procedează, de exemplu, folosind programul UNIX `dump`. Fișierele corespunzătoare tabelelor bazelor dumneavoastră de date se pot afla în flux datorită activității serverului atunci când are loc execuția copiei de siguranță a sistemului, deci restaurarea acestor fișiere nu va crea tabele consecvente din punct de vedere intern. Programul `mysqldump` creează copii de siguranță care sunt mai utile pentru restaurarea bazelor de date și vă permite să creați copii de siguranță fără oprirea serverului.

  Dacă decideți să rulați un sistem de baze de date pe un calculator gazdă mai rapid sau dacă doriți să-l replicați, va trebui să copiați conținutul acestuia pe un alt calculator. Trebuie să înțelegeți procedura de efectuare a acestei operații, în caz de nevoie. Fișierele de baze de date pot fi dependente de sistem, deci sunteți obligat să nu vă rezumați la copierea fișierelor.
- **Ajustarea serverului.** Utilizatorii dumneavoastră doresc ca serverul să funcționeze la un nivel maxim de performanțe. Metoda rapidă de îmbunătățire a modului de funcționare a serverului este de a achiziționa o cantitate superioară de memorie sau unități de disc mai rapide. Dar aceste tehnici „de forță brută" nu pot înlocui necesitatea de a înțelege modul de funcționare a serverului. Trebuie să știți care sunt parametrii disponibili pentru ajustarea modului de funcționare a serverului și modul în care aceștia se aplică în situația dumneavoastră. În unele locuri, interogările sunt orientate îndeosebi spre operații de regăsire. În alte părți, predomină inserțiile și actualizările. Opțiunea privind parametrii de modificat va fi influențată de combinația de interogări pe care o controlați la locul dumneavoastră de muncă.

- **Servere multiple.** În unele circumstanțe, rularea mai multor servere este utilă. Puteți testa o nouă versiune MySQL lăsând instalarea de producție curentă în starea în care se află, respectiv puteți asigura o mai bună confidențialitate pentru diferite grupuri de utilizatori. (Ultimul scenariu este important mai ales pentru furnizorii de servicii Internet.) În asemenea situații, trebuie să cunoașteți modul de configurare al mai multor instalări simultane.
- **Actualizări ale sistemului MySQL.** Noile versiuni ale sistemului MySQL apar frecvent. Trebuie să cunoașteți modul de a fi mereu la curent cu aceste versiuni, pentru a beneficia de remediile hibelor existente și de noile caracteristici. Este necesar să înțelegeți circumstanțele în care este mai bine să vă abțineți de la trecerea la o versiune mai recentă, respectiv să știți să alegeți între versiunile stabile și cele de dezvoltare.

Securitatea

Când rulați o instalare a sistemului MySQL, este important să vă asigurați că datele pe care le stochează utilizatorii dumneavoastră sunt păstrate în siguranță. Administratorul MySQL este responsabil cu controlul accesului la catalogul de date și la server, după cum trebuie să înțeleagă următoarele aspecte:

- **Securitatea sistemului de fișiere.** Un calculator UNIX poate găzdui numeroase conturi de utilizator, care nu au nici un fel de atribuții administrative în raport cu MySQL. Este important să vă asigurați că aceste conturi nu au acces la catalogul de date. Acest lucru le împiedică să compromită datele de la nivelul unui sistem de fișiere prin copierea sau eliminarea tabelelor din baza de date sau prin posibilitatea de a citi fișiere jurnal care pot conține anumite informații cu caracter confidențial. Trebuie să știți să configurați un cont de utilizator UNIX pentru serverul MySQL, modul de configurare a catalogului de date astfel încât să fie în proprietatea acelui utilizator, precum și modul de pornire a serverului astfel încât să ruleze cu privilegiile utilizatorului respectiv.
- **Securitatea serverului.** Trebuie să înțelegeți modul de funcționare a sistemului de securitate MySQL astfel încât, atunci când configurați conturi de utilizator, să le acordați privilegiile corespunzătoare. Utilizatorii care se conectează la server din rețea trebuie să aibă permisiunea de a executa numai acele operații pe care trebuie să le execute. Nu doriți să acordați accidental un acces de tip superutilizator unor utilizatori anonimi, datorită unei înțelegeri defectuoase a sistemului de securitate!

Întreținerea și repararea bazelor de date

Fiecare administrator MySQL speră să nu aibă de-a face cu tabele de baze de date distruse sau alterate. Speranța singură nu previne însă apariția problemelor. Există etape pe care le puteți parcurge pentru a reduce riscurile la minimum și a învăța ce aveți de făcut dacă se întâmplă lucruri mai puțin plăcute:

- **Refacerea în urma căderilor.** Dacă, în ciuda eforturilor dumneavoastră celor mai susținute, dezastrul se produce, trebuie să cunoașteți metodele de a vă repara sau restaura tabelele. Refacerea în urma căderilor este necesară numai rareori, dar, atunci când *este* necesară, este o treabă neplăcută și cu o mare doză de stres (mai ales când telefonul sună și lumea bate la ușă în timp ce dumneavoastră vă spargeți capul încercând să rezolvați problema). Cu toate acestea, trebuie să știți cum să procedați, deoarece altfel utilizatorii dumneavoastră vor fi foarte prost-dispuși! Trebuie să cunoașteți caracteristicile de verificare a tabelelor și de reparare specifice utilitarelor `isamchk`, respectiv `myisamchk`. Este necesar să știți să executați cea mai bună refacere posibilă pornind de la copiile de siguranță, precum și să folosiți jurnalele de actualizare pentru a reface modificările care s-au produs după executarea copiei de siguranță celei mai recente.
- **Întreținere preventivă.** Trebuie implementat un program sistematic de întreținere preventivă, pentru a reduce la minimum posibilitatea de alterare sau deteriorare a bazelor de date. De asemenea, trebuie să efectuați copii de siguranță, desigur, dar întreținerea preventivă reduce posibilitatea de a fi obligat să le folosiți.

Descrierea schematică furnizată anterior rezumă responsabilitățile pe care vi le asumați atunci când deveniți administrator al unui sistem MySQL. Următoarele câteva capitole discută despre acestea într-un mod mai detaliat și conțin procedurile care trebuie respectate pentru a vă putea îndeplini aceste responsabilități în mod eficient. Vom discuta mai întâi despre catalogul de date MySQL; aceasta este resursa pe care o întrețineți și este necesar să-i înțelegeți structura și conținutul. De aici, vom trece la îndatoririle administrative generale, la o discuție despre sistemul de securitate MySQL, precum și la aspecte legate de întreținere și depanare.

CAPITOLUL 10

Catalogul de date MySQL

Din punct de vedere conceptual, majoritatea sistemelor de baze de date relaționale sunt similare; toate sunt alcătuite dintr-un set de baze de date, iar fiecare bază de date include un set de tabele. Dar fiecare sistem își are propria sa modalitate de a-și organiza datele pe care le gestionează, iar MySQL nu este o excepție.

În mod prestabilit, toate datele gestionate de serverul MySQL `mysqld` sunt stocate într-o locație denumită *catalogul de date* al sistemului MySQL. Toate bazele de date sunt stocate aici, după cum tot aici se află fișierele de stare care furnizează informații despre modul de operare a serverului. Dacă executați unele operații administrative pentru o instalare a sistemului MySQL, trebuie să cunoașteți structura și modul de utilizare a catalogului de date.

Acest capitol tratează despre următoarele subiecte:

- **Modul de determinare a amplasamentului catalogului de date.** Trebuie să cunoașteți acest lucru, pentru a putea administra în mod eficient conținutul catalogului de date.
- **Modul în care serverul organizează și furnizează accesul la bazele de date și la tabelele pe care le gestionează.** Acest aspect este important pentru configurarea planificării operațiilor de întreținere preventivă, precum și pentru efectuarea refacerii în urma căderilor de sistem, în cazul în care se produc alterări ale tabelelor.
- **Unde se găsesc fișierele de stare generate de server și care este conținutul acestora.** Conținutul lor furnizează informații utile despre modul în care rulează serverul, ceea ce poate deveni util dacă întâlniți probleme.
- **Metoda de modificare a amplasamentului prestabilit sau a organizării catalogului de date sau a unor baze de date individuale.** Acest aspect poate fi important pentru gestiunea alocării resurselor de spațiu pe disc în sistemul dumneavoastră – de exemplu, prin echilibrarea activității pe disc între toate unitățile sau pentru modificarea amplasamentului datelor pe sistemele de fișiere care dispun de o cantitate mai mare de spațiu liber. Puteți folosi aceste cunoștințe și pentru planificarea amplasamentului noilor baze de date.

Puteți dobândi avantaje din lectura acestui capitol chiar dacă nu efectuați operații de administrare ale unui sistem MySQL; nu strică niciodată să cunoașteți mai bine modul de operare a serverului.

Amplasarea catalogului de date

O locație prestabilită a catalogului de date este compilată în server. Amplasările prestabilite caracteristice sunt `/usr/local/var` dacă instalați MySQL dintr-o distribuție sursă, `/usr/local/mysql/data` dacă instalați dintr-o distribuție binară, respectiv `/var/lib/mysql` dacă instalați dintr-un fișier RPM.

Amplasarea catalogului de date poate fi specificată explicit atunci când porniți serverul, folosind o opțiune `--datadir=//cale/spre/catalog`. Acest fapt este util dacă doriți să plasați catalogul de date în altă parte decât în locația sa prestabilită.

Ca administrator MySQL, trebuie să știți unde se găsește catalogul dumneavoastră de date. Dacă rulați mai multe servere, trebuie să știți unde se află toate cataloagele de date. Dar, dacă nu cunoașteți locația (poate că ați luat locul unui fost administrator ale cărui notițe lăsau de dorit), există numeroase modalități de a afla:

- **Folosiți variabilele `mysqladmin` pentru a obține calea de acces spre catalogul de date direct de la serverul dumneavoastră.** În UNIX, datele de ieșire se prezintă astfel:

```
% mysqladmin variables
```

Variable_name	Value
back_log	5
connect_timeout	5
basedir	/var/local/
datadir	/usr/local/var/
...	

Aceste date de ieșire indică o amplasare a catalogului de date în locația `/usr/local/var` din gazda serverului.

În Windows, datele de ieșire ar putea arăta astfel:

```
C:\> mysqladmin variables
```

Variable_name	Value
back_log	5
connect_timeout	5
basedir	c:\mysql\
datadir	c:\mysql\data\
...	

Dacă aveți mai multe servere care rulează, acestea vor aștepta eventualele conexiuni la diferite numere de porturi TCP/IP sau socluri. Puteți obține informații despre catalogul de date de la fiecare din aceste servere, furnizând opțiunile `--port` sau `--socket` adecvate, pentru a vă conecta la portul, respectiv soclul la care fiecare server așteaptă conexiunile cu clienții:

```
% mysqladmin --port=numar_port variables
% mysqladmin --socket=/cale/spre/soclu variables
```

Comanda `mysqladmin` poate fi rulată pe orice gazdă de la care vă puteți conecta la server. Folosiți o opțiune `--host=nume_gazda` dacă trebuie să vă conectați la server de la o gazdă aflată la distanță:


```
% mysqladmin --host=nume_gazda variables
```

În Windows, puteți contacta un server Windows NT care așteaptă conexiuni la un canal denumit folosind opțiunea `--pipe` pentru a forța o conexiune printr-un canal denumit, respectiv opțiunea `--socket=nume_canal` pentru a specifica numele canalului:

```
C:\ mysqladmin --pipe --socket=nume_canal variables
```

- **Puteți folosi `ps` pentru a vedea linia de comandă a oricărui proces `mysqld` care se execută la un moment dat.** Încercați una din următoarele comenzi (în funcție de versiunea `ps` acceptată de sistemul dumneavoastră) și căutați opțiunea `--datadir` în comenzile care sunt prezentate în datele de ieșire:

```
% ps axww | grep mysqld       Comandă ps în stil BSD
% ps -ef | grep mysqld        Comandă ps în stil System V
```

Comanda `ps` poate fi utilă mai ales dacă sistemul dumneavoastră rulează mai multe servere, deoarece puteți descoperi simultan mai multe locații ale cataloagelor de date. Dezavantajele sunt că `ps` trebuie rulat pe gazda serverului și că nu vor fi generate nici un fel de informații utile decât dacă opțiunea `--datadir` a fost specificată în mod explicit în linia de comandă `mysqld`.

- **Dacă MySQL a fost instalat de la o distribuție sursă, îi puteți examina informațiile de configurare pentru a determina amplasarea catalogului de date.** De exemplu, locația este disponibilă în catalogul `Makefile` situat la cel mai înalt nivel. Dar fiți atent: locația este valoarea variabilei `localstatedir` din `Makefile`, nu a variabilei `datadir`, așa cum vă așteptați. De asemenea, dacă distribuția este amplasată pe un sistem de fișiere montat cu NFS și este folosită pentru a construi MySQL pe numeroase sisteme gazdă, informațiile de configurare reflectă sistemul pe care distribuția a fost cel mai recent construită. Astfel, aceste informații pot să nu conțină catalogul de date pentru gazda care vă interesează.
- **Dacă nici una din metodele precedente nu dă rezultate, puteți folosi `find` pentru a căuta în fișierele din baza de date.** Comanda următoare caută fișierele `.frm` (de descriere), care fac parte din orice instalare a sistemului MySQL:

```
% find / -name "*.frm" -print
```

În exemplele care vor fi prezentate pe parcursul acestui capitol, voi nota amplasarea catalogului de date MySQL sub numele DATADIR. Trebuie să interpretați această notație ca fiind amplasarea catalogului de date în propriul dumneavoastră calculator.

Structura catalogului de date

Catalogul de date MySQL conține toate bazele de date și tabelele gestionate de server. Acestea sunt organizate într-o structură arborescentă care este implementată într-o manieră directă, utilizându-se structura ierarhică a sistemelor de fișiere UNIX sau Windows:

- Fiecare bază de date corespunde unui catalog din cadrul catalogului de date.
- Tabelele dintr-o bază de date corespund fișierelor din catalogul bazei de date.

De asemenea, catalogul de date conține numeroase fișiere de stare care sunt generate de server, precum fișierele jurnal. Aceste fișiere conțin informații importante privind operarea serverului și sunt valoroase pentru administratori, mai ales atunci când se întâm-

plă ceva grav și dumneavoastră încercați să determinați cauza problemei. Dacă o anumită interogare suprimă serverul, de exemplu, puteți identifica interogarea ostilă prin examinarea fișierelor jurnal.

Modul în care serverul MySQL furnizează accesul la date

Tot ceea ce se află sub catalogul de date este gestionat de o singură entitate: serverul MySQL `mysqld`. Programele client nu manipulează niciodată datele în mod direct. În schimb, serverul asigură unicul punct de contact prin care se obține accesul la bazele de date, comportându-se ca intermediar între programele client și datele pe care doresc să le utilizeze acestea. (Vezi figura 10.1.)

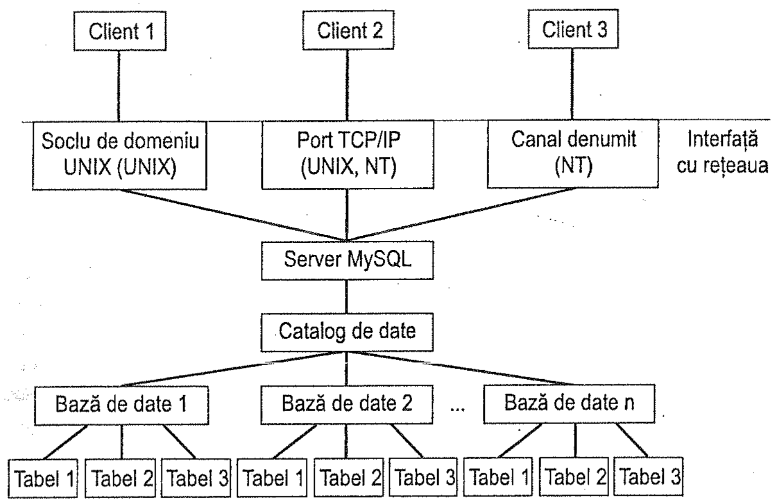


Figura 10.1 Modul în care serverul MySQL controlează accesul la catalogul de date.

Când serverul pornește, deschide fișierele jurnal, dacă a fost cerut vreunul, apoi prezintă o interfață din rețea a catalogului de date, așteptând conexiunile din rețea. Pentru a obține acces la date, programele client stabilesc o conexiune cu serverul, apoi comunică cererile sub formă de interogări SQL pentru a efectua operațiile dorite (de exemplu crearea unui tabel, selectarea înregistrărilor, actualizarea înregistrărilor). Serverul execută fiecare operație și trimite rezultatul clientului. Serverul este multi-fir și poate deservi multiple conexiuni simultane cu clientul. Totuși, deoarece operațiile de actualizare se execută una câte una, efectul practic constă în stabilirea unei succesiuni a cererilor, astfel încât doi clienți să nu poată modifica niciodată aceeași înregistrare simultan.

În condiții normale, a determina serverul să se comporte ca unic arbitru al accesului la bazele de date furnizează o asigurare împotriva categoriilor de deteriorări ce pot surveni din accesul simultan al mai multor procese la tabelele bazei de date. Cu toate acestea, administratorii trebuie să știe că există situații când serverul nu are un control exclusiv asupra catalogului de date:

- **Când rulați mai multe servere pe un singur catalog de date.** În mod normal, rulați un singur server pentru a gestiona toate bazele de date de pe un calculator gazdă, dar este posibilă rularea mai multor servere. Dacă se recurge la acest procedeu pentru a furniza accesul la mai multe cataloage de date independente, nu există nici o problemă de interacțiune. Dar este posibil să porniți mai multe servere și să le orientați spre același catalog de date. În general, aceasta nu este o idee bună. Dacă o încercați, este bine să vă asigurați că sistemul dumneavoastră furnizează o bună blocare a fișierelor; în caz contrar, serverele dumneavoastră nu vor colabora corect. De asemenea, riscați ca fișierele dumneavoastră să devină o sursă de confuzie (nu o sursă de informații utile) dacă aveți mai multe servere care scriu în aceste fișiere simultan.
- **Când rulați `isamchk` și `myisamchk`.** Utilitarele `isamchk` și `myisamchk` sunt folosite pentru întreținerea, depanarea și repararea tabelelor. Așa cum puteți intui, deoarece aceste utilitare pot modifica conținutul tabelelor, a le permite să opereze asupra tabelelor în același timp cu serverul poate determina deteriorări ale tabelelor. Este important să înțelegeți modul de limitare al acestui tip de interacțiune, astfel încât să nu vă deteriorați tabelele. Vezi capitolul 13, „Întreținerea și repararea bazelor de date", pentru instrucțiuni privind utilizarea corectă a acestor programe.

Reprezentarea bazelor de date

Fiecare bază de date gestionată de serverul MySQL își are propriul său catalog de baze de date. Acesta există ca un subcatalog al catalogului de date, care are același nume cu baza de date pe care o reprezintă. De exemplu, o bază de date `my_db` corespunde catalogului de baze de date `DATADIR/my_db`.

Această reprezentare permite o implementare aproape banală a numeroase instrucțiuni la nivel de baze de date. Instrucțiunea `CREATE DATABASE nume_baza_de_date` creează un catalog vid `nume_baza_de_date` în cadrul catalogului de date, cu un proprietar și cu un mod care permit numai accesul utilizatorului serverului MySQL (utilizatorul UNIX sub al cărui cont rulează serverul). Acest fapt este echivalent cu crearea manuală a bazelor de date prin executarea următoarelor comenzi asupra gazdei serverului, din contul utilizatorului serverului:

`% mkdir DATADIR/nume_baza_de_date`	Creează catalogul bazei de date
`% chmod 700 DATADIR/nume_baza_de_date`	Face baza de date accesibilă numai pentru utilizatorul serverului MySQL

Abordarea minimală a reprezentării unei baze de date noi prin intermediul unui catalog vid contrastează cu alte sisteme de baze de date, care creează un număr de fișiere de control sau fișiere sistem chiar și pentru o bază de date „vidă".

Instrucțiunea `DROP DATABASE` este de asemenea implementată cu ușurință. `DROP DATABASE nume_baza_de_date` elimină catalogul `nume_baza_de_date` din catalogul de date, alături de toate fișierele tabel incluse în acesta. Comanda este aproape similară cu următoarea:

```
% rm -rf DATADIR/nume_baza_de_date
```

(Diferența constă în faptul că serverul elimină numai fișiere cu extensii cunoscute ca fiind utilizate pentru tabele. Dacă ați creat alte fișiere în catalogul bazei de date, serverul le lasă intacte și catalogul nu este eliminat.)

SHOW DATABASES nu este, în esență, nimic mai mult decât o listă a numelor cataloagelor amplasate în cadrul catalogului de date. Unele sisteme de baze de date folosesc un tabel principal care conține toate bazele de date existente în sistem, dar în MySQL nu există o asemenea structură. Dată fiind simplitatea structurii catalogului de date, lista bazelor de date este implicită în conținutul catalogului de date, iar un asemenea tabel ar reprezenta o suprasarcină inutilă.

Reprezentarea tabelelor din baza de date

Fiecare tabel dintr-o bază de date există sub forma a trei fișiere din catalogul bazei de date: un fișier formular (descriere), un fișier de date și un fișier index. Numele de bază al fiecărui fișier este numele tabelului, iar extensia numelui de fișier indică tipul fișierului. Extensiile sunt prezentate în tabelul 10.1. Extensiile fișierelor de date și index indică dacă tabelul folosește sistemul de indexare mai vechi ISAM, respectiv indexarea mai recentă de tip MyISAM.

Tabelul 10.1 Tipuri de fișiere MySQL

Tipul fișierului	Extensia numelui de fișier	Conținutul fișierului
Fișier formular	`.frm`	Descrie structura tabelului (coloane, tipuri de coloane, indexuri etc.)
Fișier de date	`.ISD` (ISAM) sau `.MYD` (MyISAM)	Conține datele din tabel – adică rândurile sale.
Fișier index	`.ISM` (ISAM) sau `.MYI` (MyISAM)	Conține arborii de index pentru toate indexurile din fișierul de date. Fișierul index există indiferent dacă tabelul are sau nu indexuri.

Când emiteți o instrucțiune CREATE TABLE nume_tabel care definește o structură a unui tabel, serverul creează un fișier nume_tabel.frm, care conține codificarea internă a acelei structuri. De asemenea, creează fișiere de date și index vide, inițializate cu informații care să indice inexistența înregistrărilor și a indexurilor. (Dacă instrucțiunea CREATE TABLE include specificații de index, fișierul index reflectă aceste indexuri.) Proprietatea și modul fișierelor care reprezintă tabelul sunt configurate astfel încât să permită accesul numai utilizatorului serverului MySQL.

Când emiteți o instrucțiune ALTER TABLE, serverul codifică din nou fișierul nume_tabel.frm și operează modificări în conținutul fișierelor de date și al fișierelor index, astfel încât acestea să reflecte schimbarea structurală indicată de instrucțiune. Acest lucru este valabil și pentru instrucțiunile CREATE INDEX și DROP INDEX, deoarece serverul le tratează ca instrucțiuni echivalente cu ALTER TABLE. Instrucțiunea DROP TABLE este implementată prin eliminarea celor trei fișiere care reprezintă tabelul.

Nu puteți crea sau modifica manual un tabel, deși puteți elimina un tabel prin eliminarea celor trei fișiere din catalogul bazei de date căreia îi corespunde tabelul. De exemplu, dacă my_db este baza de date curentă, instrucțiunea DROP TABLE tabelul_meu este aproximativ echivalentă cu următoarea comandă:

```
% rm -f DATADIR/my_db/tabelul_meu.*
```


Rezultatul comenzii SHOW TABLES my_db se rezumă la un listing al numelor de bază ale fișierelor .frm din catalogul my_db. Unele sisteme de baze de date folosesc un registru care conține toate tabelele dintr-o bază de date. MySQL nu procedează astfel, deoarece nu este necesar; „registrul" există implicit în structura catalogului de date.

Restricții ale sistemului de operare asupra denumirii bazelor de date și tabelelor

MySQL are reguli generale pentru denumirea bazelor de date și a tabelelor:

- Numele pot fi construite din caractere alfanumerice din setul de caractere curent, precum și din caracterele de subliniere și dolar (_ și $).
- Numele pot avea o lungime de maximum 64 de caractere.

Totuși, deoarece numele bazelor de date și ale tabelelor corespund numelor cataloagelor și ale fișierelor, sistemul de operare pe care rulează un server poate impune restricții suplimentare.

Mai întâi, în numele bazelor de date și ale tabelelor sunteți obligat să folosiți doar caracterele permise în numele fișierelor. De exemplu, caracterul $ este permis într-un nume de către regulile sistemului MySQL, dar, dacă sistemul dumneavoastră de operare nu-l permite, atunci nu-l veți putea folosi nici în numele cataloagelor sau al tabelelor. În practică, aceasta nu este o problemă nici pentru UNIX, nici pentru Windows. Cea mai mare dificultate pe care o puteți avea este referirea la nume direct din interpretor atunci când efectuați operații de administrare a bazei de date. De exemplu, dacă atribuiți unei baze de date un nume precum $my_db, care include simbolul dolarului, orice referință la numele respectiv dintr-o linie de comandă a interpretorului poate fi interpretată de acesta ca o referință la o variabilă:

```
% ls $my_db
my_db: Undefined variable. (variabilă nedefinită)
```

Dacă se întâmplă acest lucru, trebuie să modificați semnificația caracterului $ sau să folosiți ghilimelele pentru a elimina semnificația specială a acestuia:

```
% ls \$my_db
% ls '$my_db'
```

Dacă folosiți ghilimele, utilizați-le pe cele simple. Ghilimelele duble nu inhibă interpretarea numelui drept variabilă.

În al doilea rând, deoarece MySQL permite ca numele bazelor de date și ale tabelelor să aibă o lungime de maximum 64 de caractere, lungimea numelor este de asemenea asociată cu lungimea permisă de sistemul dumneavoastră de operare. În mod normal, aceasta nu este o problemă, deși sub UNIX puteți nimeri peste sisteme de tip System V, care impun o limită mai veche, de 14 caractere. În acest caz, limita efectivă a numelor bazelor de date este de 14 caractere. Limita pentru numele tabelelor este de 10 caractere, deoarece numele fișierelor care reprezintă tabelele se încheie cu un punct și cu o extensie formată din trei caractere.

În al treilea rând, sensibilitatea la diferența între majuscule și minuscule manifestată de sistemul de fișiere de bază influențează modul de denumire la bazele de date și la tabele, precum și modul de referire la acestea. Dacă sistemul de fișiere este sensibil la diferența

între majuscule și minuscule (ca în UNIX), numele `tabelul_meu` și `TABELUL_MEU` se referă la două tabele diferite. Dacă sistemul de fișiere nu este sensibil la diferența între majuscule și minuscule (ca în Windows), numele `tabelul_meu` și `TABELUL_MEU` se referă la unul și același tabel. Trebuie să rețineți acest aspect atunci când folosiți un server UNIX pentru a dezvolta o bază de date, în cazul în care este posibil să mutați baza de date pe un server Windows.

Implicațiile structurii catalogului de date asupra performanțelor sistemului

Structura catalogului de date este ușor de înțeles, deoarece folosește într-un mod natural structura ierarhică a sistemului de fișiere. În același timp, această structură are anumite implicații de performanță, mai ales în ceea ce privește operațiile care deschid fișierele ce reprezintă tabelele din bazele de date.

O consecință a structurii catalogului de date este aceea că, deoarece tabelele sunt reprezentate de mai multe fișiere, fiecare tabel deschis poate necesita mai mulți descriptori de fișiere, nu unul singur. Serverul memorează în mod inteligent descriptorii în cache, dar un server solicitat poate folosi cu ușurință un număr foarte mare de asemenea descriptori, deservind numeroase conexiuni client simultane sau executând interogări complexe, care fac referire la multe tabele. Descriptorii de fișiere constituie o resursă prețioasă în numeroase sisteme, mai ales în cele unde limita prestabilită per proces este relativ redusă. Capitolul 11, „Administrarea generală a sistemului MySQL“, conține informații despre estimarea numărului de descriptori de care veți avea nevoie, precum și despre reconfigurarea serverului sau a sistemului de operare, dacă este necesar.

Un alt efect al reprezentării fiecărui tabel prin propriile sale fișiere este că timpul necesar deschiderii unui tabel crește cu numărul de tabele. Operațiile de deschidere a tabelelor se mapează pe operațiile de deschidere a fișierelor furnizate de sistemul de operare, fiind ca atare dependente de eficiența rutinelor de căutare în cataloage ale sistemului. În mod normal, aceasta nu reprezintă o problemă, dar este ceva de care trebuie ținut cont dacă aveți nevoie de numeroase tabele într-o bază de date.

De exemplu, dacă doriți să aveți 10000 de tabele, catalogul bazei de date va conține 30000 de fișiere. Cu un număr atât de mare de fișiere, veți remarca o scădere a performanțelor, datorită timpului necesar operațiilor de deschidere a fișierelor. (Sistemele de fișiere Linux ext2 și Solaris prezintă această problemă.) Dacă viteza de deschidere a fișierelor reprezintă o problemă, este indicat să reconsiderați structura tabelelor dumneavoastră în raport cu necesitățile aplicațiilor și să vă reorganizați tabelele în consecință. Întrebați-vă dacă aveți într-adevăr nevoie de un număr atât de mare de tabele: uneori, aplicațiile măresc în mod inutil numărul de tabele. O aplicație care creează un tabel separat pentru fiecare utilizator generează numeroase tabele, toate cu structuri identice. Dacă doriți să combinați tabelele într-un singur tabel, puteți proceda astfel prin adăugarea unei alte coloane, care să identifice utilizatorul căruia îi aparține fiecare rând. Dacă astfel se reduce în mod semnificativ numărul de tabele, performanțele sistemului se îmbunătățesc.

Ca în orice proiectare a unei baze de date, trebuie să vă gândiți dacă această strategie merită sau nu să fie utilizată într-o aplicație dată. Motivele pentru a nu combina tabelele în maniera descrisă mai sus sunt următoarele:

- **Necesități sporite privind spațiul pe disc.** Prin combinarea tabelelor se reduce numărul de tabele necesar (scăderea timpilor de deschidere a unui tabel) dar se adaugă încă o coloană (necesități sporite de spațiu pe disc). Acesta este un compromis caracteristic între timp și spațiu, iar dumneavoastră va trebui să decideți care este factorul cel mai important. Dacă viteza este de o importanță capitală, probabil că veți dori să sacrificați puțin spațiu suplimentar pe disc. Dacă spațiul este limitat, este de preferat să folosiți mai multe tabele și să tolerați ușoare întârzieri.
- **Considerații legate de securitate.** Acestea pot limita capacitatea sau dorința dumneavoastră de combinare a tabelelor. Un motiv de a utiliza un tabel separat pentru fiecare utilizator este de a permite accesul la tabelul respectiv numai utilizatorului respectiv, prin intermediul unor privilegii la nivel de tabel. Dacă veți combina tabelele, datele tuturor utilizatorilor se vor găsi în același tabel.

MySQL nu prevede nici o măsură pentru a limita accesul unui utilizator dat la mai multe rânduri; astfel, s-ar putea să nu reușiți să fuzionați tabele fără a compromite controlul accesului. Pe de altă parte, dacă întregul acces la date este controlat de aplicația dumneavoastră (utilizatorii nu se conectează niciodată în mod direct la baza de date), puteți fuziona tabelele și folosi logica de aplicație pentru a aplica rezultatului combinat regulile de acces la nivel de rând.

MySQL își are propria limită internă în ceea ce privește dimensiunile tabelelor, dar, deoarece reprezintă tabelele sub formă de fișiere, este de asemenea limitat de dimensiunea maximă a fișierelor permisă de sistemul dumneavoastră de operare. Ca atare, dimensiunea maximă a unui tabel este, ca atare, cea mai mică valoare dintre limita internă a sistemului MySQL și limita dimensiunilor fișierelor.

În general, se tinde spre o relaxare în timp a restricțiilor privind dimensiunile. De exemplu, IBM AIX 4.1 are o limită a dimensiunilor fișierelor situată la 4,1 GB, în timp ce limita sistemului AIX 4.2 este de aproximativ 64GB. Limita internă a dimensiunilor tabelelor în MySQL crește de la o versiune la alta. Anterior seriei 3.23, limita internă era de 4GB. Începând cu versiunea 3.23, limita este de aproximativ 9 milioane de teraocteți. Tabelul 10.2 ilustrează modul în care interacțiunea dintre limita internă a dimensiunii unui tabel în MySQL și limita dimensiunii unui fișier în AIX determină dimensiunea maximă efectivă a unui tabel. Interacțiuni similare se pot aplica și altor sisteme de operare.

Tabelul 10.2 Interacțiunea între limitele dimensionale impuse de MySQL și sistemul de operare

Versiune MySQL	Versiune AIX	Dimensiunea maximă a tabelului	Factor limitativ
MySQL 3.22.22	AIX 4.1	2GB	Dimensiunea maximă a fișierului de 2GB impusă de AIX

Continuare

Tabelul 10.2 Continuare

Versiune MySQL	Versiune AIX	Dimensiunea maximă a tabelului	Factor limitativ
MySQL 3.22.22	AIX 4.2	4GB	Dimensiunea maximă a fişierului de 4GB impusă de MySQL
MySQL 3.23.0	AIX 4.1	2GB	Dimensiunea maximă a fişierului de 2GB impusă de AIX
MySQL 3.23.0	AIX 4.26	4GB	Dimensiunea maximă a fişierului de 64GB impusă de AIX

Fişierele de stare din MySQL

În afară de cataloagele bazelor de date, catalogul de date din MySQL conţine un număr de fişiere de stare. Aceste fişiere sunt rezumate în tabelul 10.3, iar apoi vor fi descrise mai detaliat. Numele prestabilit pentru majoritatea fişierelor este generat pornind de la numele gazdei serverului, simbolizat în tabel prin notaţia NUMEGAZDA.

Serverul scrie identificatorul său de proces (PID) în fişierul PID atunci când îşi începe execuţia, respectiv elimină fişierul atunci când se opreşte. Fişierul PID este mijlocul prin care serverul permite localizarea sa de către alte procese. De exemplu, dacă rulaţi scriptul `mysql.server` la momentul închiderii sistemului pentru a închide serverul MySQL, acel script examinează fişierul PID pentru a determina procesul căruia trebuie să-i trimită un semnal de terminare.

Jurnalul de eroare este creat de `safe_mysqld`, ca o redirecţionare a ieşirii standard în caz de eroare a serverului; conţine toate mesajele scrise de server la `stderr`. Aceasta înseamnă că jurnalul de eroare există numai dacă porniţi serverul invocând `safe_mysqld`. (Aceasta este metoda preferată pentru pornirea serverului indiferent de situaţie, deoarece `safe_mysqld` reporneşte serverul, dacă acesta îşi încheie execuţia datorită unei erori.)

Fişierul jurnal general şi fişierul de actualizare sunt opţionale; puteţi activa numai tipurile de jurnale dorite, folosind opţiunile serverului `--logg` şi `--log-update`.

Tabelul 10.3 Fişierele de stare MySQL

Tip de fişier	Nume prestabilit	Conţinutul fişierului
Identificator de proces	NUMEGAZDA.`pid`	Identificatorul de proces al serverului
Jurnal de eroare	NUMEGAZDA.`err`	Evenimentele de pornire şi oprire, respectiv condiţiile de eroare
Jurnal general	NUMEGAZDA.`log`	Evenimentele de conectare-deconectare şi informaţii privind interogările
Jurnal de actualizare	NUMEGAZDA.`nnn`	Textele tuturor interogărilor care modifică structura sau conţinutul tabelelor

Jurnalul general furnizează informaţii generale despre operarea serverului, în speţă identitatea celui care se conectează, locul de unde se conectează şi interogările pe care le emite. Jurnalul de actualizare furnizează de asemenea informaţii despre interogări, dar numai pentru interogările care aduc modificări conţinutului bazei de date. Conţinutul jurnalului de actualizare este scris sub formă de instrucţiuni SQL care pot fi executate furnizându-le ca date de intrare pentru clientul `mysql`. Jurnalele de actualizare sunt utile dacă sistemul a avut o cădere şi trebuie să reveniţi la fişierele care conţin copiile de siguranţă, deoarece puteţi repeta actualizările efectuate de la momentul căderii, alimentând serverul cu jurnalele de actualizare. Astfel, veţi putea aduce bazele de date la starea în care se găseau la momentul căderii.

Iată un exemplu privind informaţiile care apar în jurnalul general ca rezultat al unei scurte sesiuni client care creează un tabel în baza de date `test`, inserează un rând în tabel şi apoi şterge tabelul:

```
990509    7:34:09   492    Connect       paul@localhost on test
                    492    Query         show databases
                    492    Query         show tables
                    492    Field List    tbl_1
                    492    Field List    tbl_2
                    ...
990509    7:34:22   492    Query    CREATE TABLE my_tbl (val INT)
990509    7:34:34   492    Query    INSERT INTO my_tbl VALUES(1)
990509    7:34:38   492    Query    DROP TABLE my_tbl
990509    7:34:40   492    Quit
```

Jurnalul general conţine coloane pentru dată şi oră, identificatorul firului serverului, tipul evenimentului şi informaţii specifice evenimentului.

În jurnalul de actualizare, aceeaşi sesiune se prezintă astfel:

```
use test:
CREATE TABLE my_tbl (val INT);
INSERT INTO my_tbl VALUES(1);
DROP TABLE my_tbl;
```

Pentru jurnalul de actualizare, este disponibilă o formă extinsă de consemnare a informaţiilor folosind opţiunea `--log-long-format`. Caracteristica de consemnare detaliată furnizează informaţii cu privire la emitentul fiecărei interogări şi momentul emiterii acesteia. Astfel, se consumă o cantitate mai mare de spaţiu pe disc, desigur, dar procedeul este util dacă doriţi să ştiţi cine ce face, fără a încerca să corelaţi conţinutul jurnalului de actualizare cu evenimentele de conexiune din jurnalul general.

Pentru sesiunea prezentată anterior, caracteristica de consemnare detaliată a informaţiilor în jurnal furnizează următoarele date:

```
# Time: 990509 7:43:42
#User@Host: paul [paul] @ localhost []
use test;
CREATE TABLE my_tbl (val INT);
#User@Host: paul [paul] @ localhost []
```

Continuare

Continuare

```
INSERT INTO my_tbl VALUES(1);
# Time: 990509 7:43:43
#User@Host: paul [paul] @ localhost []
DROP TABLE my_tbl;
```

Este bine să vă asigurați că fișierele dumneavoastră jurnal se află în siguranță și că nu pot fi citite de utilizatori arbitrari. Atât jurnalul general, cât și jurnalul de actualizare pot conține informații importante, precum parolele, deoarece includ textele interogărilor. Iată tipul de consemnare în jurnal care nu trebuie să fie citit de nimeni, deoarece afișează parola utilizatorului rădăcină:

```
990509  7:47:24 4 Query    UPDATE user SET Password=PASSWORD("secret")
                           WHERE user= "root"
```

Pentru informații referitoare la verificarea și configurarea permisiunilor aferente catalogului dumneavoastră de date, consultați capitolul 12, „Securitate". Instrucțiunile scurte pentru asigurarea catalogului de date sunt compuse din următoarea comandă:

```
% chmod 700 DATADIR
```

Rulați această comandă ca utilizator UNIX care este proprietarul catalogului de date. Verificați, de asemenea, că serverul rulează sub contul aceluiași utilizator; în caz contrar, comanda va împiedica nu numai accesul altor persoane la catalogul de date (ceea ce este de dorit), dar va opri și accesul serverului la bazele dumneavoastră de date (ceea ce nu este de dorit!).

Fișierele de stare apar la nivelul cel mai de sus al catalogului de date, ca și cataloagele bazei de date, deci probabil că vă întrebați dacă numele acestor fișiere pot intra în conflict cu numele bazelor de date sau dacă pot fi confundate cu acestea (de exemplu când serverul execută o instrucțiune `SHOW DATABASES`). Răspunsul este negativ. Informațiile de stare și cele din jurnale sunt stocate în fișiere, iar bazele de date sunt cataloagele, deci programele executabile pot face diferența între ele printr-un simplu apel la funcția `stat()`. (În acest mod le deosebește serverul.) Dacă parcurgeți personal catalogul de date, puteți face diferența între fișierele de stare și cataloagele bazelor de date folosind comanda `ls -l` și examinând primul caracter al informației de mod, pentru a vedea dacă este un caracter `-` sau un caracter `d`:

```
% ls -l DATADIR
total 31
drwxrwx---  1mysqladm  mysqlgrp  1024  May   8 13:22  bigdb
drwxrwx---  2mysqladm  mysqlgrp  1024  Dec  15 22:34  mysql
-rw-rw----  1mysqladm  mysqlgrp  69    May   9 20:11  pit-viper.001
-rw-rw-r--  1mysqladm  mysqlgrp  24168 May   9 20:11  pit-viper.err
-rw-rw----  1mysqladm  mysqlgrp  4376  May   9 20:11  pit-viper.log
-rw-r--r--  1mysqladm  mysqlgrp  5     May   9 20:11  pit-viper.pid
drwxrwx---  7mysqladm  mysqlgrp  512   Sep  10 1998   sql-bench
drwxrwx---  2mysqladm  mysqlgrp  512   May   9 07:44  test
```

De asemenea, puteți face diferența examinând numele: toate numele fișierelor de stare conțin un punct, caracter care nu se regăsește în numele cataloagelor bazelor de date (punctul nu este un caracter admis într-un nume de bază de date).

Pentru informații cu privire la tehnicile de întreținere și rotație a fișierelor jurnal, vezi capitolul 11.

Modificarea amplasamentului catalogului de date

În secțiunea anterioară s-a discutat despre structura catalogului de date în configurația sa prestabilită. Toate bazele de date și fișierele de stare sunt incluse în acesta. Totuși, aveți o oarecare libertate în determinarea amplasării conținutului catalogului de date. Această secțiune discută motivele care determină mutarea unor componente ale catalogului de date (sau chiar a catalogului în totalitate), componentele care pot fi mutate și modalitatea de efectuare a acestor schimbări.

MySQL vă permite să modificați amplasamentul catalogului de date sau a elementelor acestuia. Există numeroase motive care ar justifica o atare operație:

- Puteți amplasa catalogul de date pe un sistem de fișiere cu o capacitate superioară sistemului de fișiere unde este situat acesta în mod prestabilit.
- În cazul în care catalogul dumneavoastră de date se află pe un disc solicitat, îl puteți amplasa pe o unitate mai puțin activă, pentru a stabili un echilibru între activitatea cu discul a diferitelor unități fizice. Puteți amplasa bazele de date și fișierele jurnal în unități separate sau puteți distribui bazele de date la mai multe unități, din motive similare.
- Puteți dori să utilizați mai multe servere, fiecare cu propriul său catalog de date. Aceasta constituie o modalitate de rezolvare a problemelor legate de limitele impuse numărului descriptorilor de fișiere per proces, mai ales dacă nu puteți reconfigura nucleul sistemului dumneavoastră astfel încât să permită limite superioare.
- Unele sisteme păstrează fișierele PID ale serverului într-un catalog precum `/var/run`. Puteți dori să amplasați acolo și fișierul PID MySQL, pentru consecvența operării sistemului.

Metode de modificare a amplasamentului

Există două metode de modificare a amplasamentului elementelor din catalogul de date:

- Puteți specifica o opțiune la momentul pornirii serverului, fie în linia de comandă, fie în grupul `[mysqld]` al unui fișier cu opțiuni.
- Puteți muta entitatea al cărei amplasament va fi modificat, apoi creați o legătură simbolică (în original *symlink*) în locația originală, care indică spre noua locație.

Nici una dintre cele două metode nu este „universală" pentru toate elementele a căror amplasare o puteți modifica. Tabelul 10.4 rezumă entitățile care pot fi deplasate și metodele de modificare a amplasamentului care pot fi folosite. Dacă folosiți un fișier cu opțiuni, este posibilă specificarea opțiunilor în fișierul cu opțiuni global `/etc/my.cnf` (`C:\my.cnf` în Windows). De asemenea, versiunea Windows curentă caută și în catalogul sistem (`C:\windows` sau `C:\NT`).

Tabelul 10.4 Rezumat al metodelor de modificare a amplasamentului

Entitate al cărei amplasament trebuie modificat	Metode de mutare aplicabile
Întregul catalog de date	Opțiune la pornirea serverului sau legătură simbolică
Cataloage individuale de baze de date	Legătură simbolică
Tabele individuale din baze de date	Legătură simbolică
Fișier PID	Opțiune la pornirea serverului
Fișier jurnal general	Opțiune la pornirea serverului
Fișier jurnal de actualizare	Opțiune la pornirea serverului

De asemenea, puteți folosi fișierul cu opțiuni `my.cnf` din catalogul de date prestabilit (catalogul compilat în server). Nu recomand utilizarea acestui fișier. Dacă doriți să modificați chiar amplasamentul catalogului de date, trebuie să lăsați catalogul de date prestabilit intact, pentru a putea plasa în el un fișier cu opțiuni, care să specifice locul unde serverul va găsi catalogul de date „autentic"! Derutant. Dacă doriți să folosiți un fișier cu opțiuni pentru a specifica opțiunile serverului, cel mai bine este să folosiți fișierul `/etc/my.cnf`.

Evaluarea efectului operației de modificare a amplasamentului

Înainte de a încerca să modificați amplasamentul unei entități, este bine să verificați dacă operația va avea efectul scontat. Eu sunt un partizan al comenzilor `du`, `df` și `ls -l` pentru obținerea de informații legate de spațiul de pe disc, dar toate aceste comenzi depind de înțelegerea corectă a structurii sistemului dumneavoastră de fișiere.

Exemplul următor ilustrează o capcană subtilă care trebuie evitată la estimarea efectului operației de modificare a amplasamentului unui catalog de date. Să presupunem că acest catalog de date se află în locația `/usr/local/var` și că doriți să-l deplasați spre `/var/mysql`, deoarece `df` arată că sistemul de fișiere `/var` dispune de o cantitate mai mare de spațiu, așa cum se poate vedea în exemplul următor:

```
% df /usr /var
Filesystem   1K-blocks   Used      Avail    Capacity   Mounted on
/dev/wd0s3e  396895      292126    73018    80%        /usr
/dev/wd0s3f  1189359     1111924   162287   15%        /var
```

Cât spațiu se va elibera în sistemul de fișiere `/usr` prin modificarea amplasamentului catalogului de date? Pentru a afla, folosiți comanda `du -s` pentru a afla cantitatea de spațiu folosită de catalogul respectiv:

```
% cd /usr/local/var
% du -s
133426 .
```

Asta înseamnă aproximativ 130MB, cantitate semnificativă pentru `/usr`. Chiar așa? Să încercăm comanda `df` în catalogul de date:

```
% df /usr/local/var
Filesystem   1K-blocks     Used      Avail    Capacity   Mounted on/
dev/wd0s3f   1189359       1111924   162287   15%        /var
```


Ciudat. Dacă am cerut cantitatea de spațiu liber pentru sistemul de fișiere care conține catalogul `/usr/local/var`, de ce comanda `df` raportează spațiul liber din sistemul de fișiere `var`? Comanda `ls -l` furnizează răspunsul:

```
% ls -l /usr/local
...
lrwxrwxr-x   1  root  wheel   10 Dec  11 23:46 var -> /var/mysql
...
```

Aceste date de ieșire arată că `/usr/local/var` este o legătură simbolică spre `/var/mysql`. Cu alte cuvinte, catalogul de date *a fost deja mutat* în sistemul de fișiere `/var` și a fost înlocuit cu o legătură simbolică orientată în direcția respectivă. Cât despre eliberarea unei mari cantități de spațiu în sistemul `/usr` prin mutarea catalogului de date în sistemul de fișiere `/var`, adio și n-am cuvinte!

Morala: câteva minute alocate estimării efectului modificării amplasamentului unei entități este o investiție profitabilă. Nu durează mult și vă împiedică să pierdeți o mulțime de timp mutând lucrurile de colo-colo, pentru ca apoi să descoperiți că nu v-ați îndeplinit obiectivul.

Modificarea amplasamentului catalogului de date

Pentru a modifica amplasamentul catalogului de date, opriți serverul și mutați catalogul de date în noua sa locație. Apoi, trebuie fie să eliminați catalogul de date original și să-l înlocuiți cu o legătură simbolică orientată spre noua locație, fie să reporniți serverul cu o opțiune care indică în mod explicit spre noua locație. Tabelul 10.5 ilustrează sintaxa liniei de comandă și a fișierului cu opțiuni pentru specificarea locației.

Modificarea amplasamentului bazelor de date

Bazele de date pot fi mutate numai cu ajutorul legăturii simbolice. Pentru a modifica amplasamentul unei baze de date, opriți serverul și mutați catalogul bazei de date. Eliminați catalogul original al bazei de date, înlocuiți-l cu o legătură simbolică orientată spre noua locație și apoi reporniți serverul.

Exemplul următor prezintă modul în care puteți muta o bază de date denumită `bigdb` într-o altă locație:

```
% mysqladmin -u root -p shutdown
Enter password: ********
% cd DATADIR
% tar cf - bigdb | (cd /var/db; tar xf -)
% mv bigdb bigdb.orig
% ln -s /var/db/bigdb .
% safe_mysqld &
```

Tabelul 10.5 Sintaxa de modificare a amplasamentului catalogului de date

Sursă de opțiuni	Sintaxă
Linie de comandă	`--datadir=//cale/spre/catalog`
Fișier cu opțiuni	`[mysqld]` `datadir=/cale/spre/catalog`

Măsuri de precauție la modificarea amplasamentului

Înainte de a executa orice operație de modificare a amplasamentului, serverul trebuie oprit; după finalizarea operației, serverul va fi repornit. Pentru anumite tipuri de operații de modificare a amplasamentului, este posibil (deși nu se recomandă) ca serverul să fie menținut în funcțiune. Dacă procedați astfel, trebuie să vă asigurați că serverul nu obține accesul la baza de date pe care o mutați. De asemenea, nu uitați să emiteți o instrucțiune `FLUSH TABLE` înainte de a muta baza de date, pentru a vă asigura că serverul închide toate fișierele tabel deschise. Dacă nu luați aceste măsuri de precauție, veți deteriora tabelele.

Trebuie să executați aceste comenzi în calitate de proprietar al catalogului de date. Catalogul original al bazei de date a primit numele nou de `bigdb.orig` din motive de siguranță. După ce v-ați asigurat că serverul funcționează corect cu baza de date al cărei amplasament a fost modificat, puteți șterge baza de date originală:

```
% rm -rf bigdb.orig
```

Modificarea amplasamentului tabelelor din bazele de date

Modificarea amplasamentului tabelelor individuale nu este o idee bună. Puteți executa această operație prin mutarea fișierelor unui tabel într-o altă locație și prin crearea în catalogul bazei de date a unor legături simbolice care să indice spre fișierele respective. Totuși, dacă emiteți o instrucțiune `ALTER TABLE` sau `OPTIMIZE TABLE`, modificările dumneavoastră vor fi anulate.

Fiecare instrucțiune operează prin crearea unui tabel temporar în catalogul bazei de date care implementează modificarea sau optimizarea dumneavoastră, iar apoi prin ștergerea tabelului original și redenumirea tabelului temporar cu numele original. Rezultatul este că legăturile dumneavoastră simbolice sunt eliminate și noul tabel ajunge direct în catalogul bazei de date, acolo unde se găsea tabelul original înainte de a fi mutat. Mai departe, vechile fișiere tabel pe care le-ați mutat în afara catalogului de baze de date continuă să se găsească în locația unde le-ați mutat, iar dumneavoastră puteți să nici nu sesizați că se află acolo, ocupând spațiu pe disc. De asemenea, legăturile simbolice au fost distruse, deci, când sesizați mai târziu ce anume s-a întâmplat, s-ar putea să nu mai aveți la dispoziție nici o modalitate viabilă de a depista amplasamentul fișierelor, dacă ați uitat unde le-ați mutat.

Deoarece este dificil de garantat faptul că nici o persoană care are acces la tabel nu va modifica sau optimiza tabelul (anulând astfel orice încercare de modificare a amplasamentului), cel mai indicat este să lăsați tabelele în catalogul bazei de date.

Modificarea amplasamentului fișierelor de stare

Puteți modifica amplasamentul fișierului PID, al jurnalului general și al jurnalului de actualizare, folosind opțiunile de pornire. Jurnalul de eroare este creat de `safe_mysqld`, iar amplasamentul său nu poate fi modificat (decât dacă editați `safe_mysqld`).

Pentru a scrie un fișier de stare într-o altă locație, opriți serverul și apoi reporniți-l cu opțiunea corespunzătoare care să specifice noul amplasament al fișierului de stare. Tabelul 10.6 prezintă linia de comandă și sintaxa folosită în fișierul cu opțiuni pentru fiecare fișier.

Eliminarea unei baze de date cu amplasament modificat

Puteți elimina o bază de date cu instrucțiunea `DROP DATABASE`, dar versiunile mai vechi de MySQL au probleme la eliminarea unei baze de date al cărei amplasament a fost modificat. Tabelele din baza de date sunt eliminate corect, dar se produce o eroare atunci când serverul încearcă să elimine catalogul de baze de date, deoarece „catalogul" este de fapt o legătură simbolică, nu un catalog real. Administratorul MySQL trebuie să elimine manual catalogul de baze de date și legătura simbolică orientată spre acesta. Această problemă a fost remediată începând cu seria MySQL 3.23.

Tabelul 10.6 Sintaxa de modificare a amplasamentului fișierului de stare

Sursă de opțiuni	Sintaxă
Linie de comandă	`--pid-file=`*`fisierpid`* `--log=`*`fisierjurnal`* `--log-update=`*`fisieractualizare`*
Fișier cu opțiuni	`[mysqld]` `pid-file=`*`fisierpid`* `log=`*`fisierjurnal`* `log-update=`*`fisieractualizare`*

Dacă specificați numele unui fișier de stare sub formă de cale de acces absolută, fișierul este creat folosind calea de acces respectivă. În caz contrar, fișierul este creat în catalogul de date. De exemplu, dacă specificați `--pid-file=/var/run/mysqld.pid`, fișierul PID este `/var/run/mysqld.pid`. Dacă specificați `--pid-file=mysqld.pid`, fișierul PID este `DATADIR/mysqld.pid`.

Dacă specificați un nume de jurnal de actualizare care nu are nici o extensie, MySQL generează nume secvențiale la fiecare deschidere a jurnalului de actualizare. Aceste nume sunt create cu extensia `.nnn`, unde `.nnn` este primul număr care nu este deja folosit de către un fișier jurnal de actualizare existent (de exemplu `update.001`, `update.002` și așa mai departe). Puteți anula generarea secvențială a numelor prin specificarea unui nume care conține o extensie explicită. Apoi, serverul va folosi numai numele pe care l-ați specificat.

CAPITOLUL 11

Administrarea generală a sistemului MySQL

În acest capitol se discută despre atribuțiile pe care un administrator MySQL trebuie să le îndeplinească pentru a asigura o funcționare corectă a sistemului MySQL. Aceste responsabilități includ: asigurarea unui timp de funcționare cât mai îndelungat al serverului, configurarea unor conturi de utilizator astfel încât clienții să poată avea acces la server, întreținerea fișierelor jurnal și executarea copiilor de siguranță ale bazelor de date. De asemenea, administratorul poate exploata mai multe servere sau poate modifica parametrii de operare ai serverului, pentru a obține performanțe superioare. În final, deoarece MySQL se află într-un proces de evoluție activă, administratorul trebuie să poată determina momentul unei actualizări a sistemului MySQL prin instalarea de versiuni noi. Alte probleme administrative importante sunt prezentate în capitolele 12, „Securitate", respectiv 13, „Întreținerea și repararea bazelor de date".

În aceste capitole vor fi descrise numeroase programe care sunt utile pentru administratorii sistemelor MySQL:

- `mysqladmin` execută diferite operații de natură administrativă.
- `safe_mysqld` și `mysql.server` sunt scripturi pentru pornirea serverului MySQL `mysqld`.
- `mysqldump` este folosit la operațiile de executare a copiilor obișnuite și de siguranță ale bazelor de date.
- `myisamchk` și `isamchk` sunt folosite pentru operațiile de reparare și verificare a integrității tabelelor.

Informații suplimentare despre majoritatea acestor programe sunt prezentate în Anexa E, „Referință de programe MySQL".

Asigurarea unei noi instalări a sistemului MySQL

Poate citiți acest capitol fiindcă tocmai ați instalat sistemul MySQL în conformitate cu instrucțiunile din Anexa A, „Obținerea și instalarea programelor". În acest caz, trebuie să configurați parolele pentru utilizatorul `root` din MySQL; într-o instalare nouă a sistemului MySQL, serverul rulează cu privilegii care nu sunt sigure. Presupun că v-ați inițializat catalogul de date și baza de date `mysql` care conține tabelele de acordare. Sub UNIX, această operație se execută prin rularea scriptului `mysql_install_db`. Sub Windows, catalogul de date și baza de date `mysql` sunt inițializate prin rularea programului `Setup` din distribuția server. De asemenea, voi presupune că serverul este în funcțiune.

Când instalați sistemul MySQL pentru prima dată, privilegiile din tabelele de acordare pentru baza de date `mysql` sunt inițializate după cum urmează:

- Vă puteți conecta ca `root` de la gazda locală *fără parolă*. Utilizatorul `root` are toate privilegiile (inclusiv privilegiile administrative) și poate executa orice operație. (Că veni vorba, faptul că superutilizatorul din MySQL și omologul său din UNIX au ambii numele de `root` este pură coincidență. Cei doi nu au nici o legătură unul cu altul.)
- Utilizatorilor care se conectează de la gazda locală le este permis accesul anonim la baza de date denumită `test` sau la toate bazele de date ale căror nume încep cu particula `test_`. Utilizatorii anonimi pot folosi în orice mod aceste baze de date, dar nu au nici un privilegiu administrativ.

Conexiunile cu serverul de la gazda locală sunt permise indiferent dacă utilizatorul care se conectează specifică numele de gazdă `localhost` sau numele real al gazdei. De exemplu, dacă serverul se află pe calculatorul `pit-viper.snake.net`, un client al gazdei respective se poate conecta la server în absența unei parole, pentru a folosi baza de date `test`, cu oricare dintre aceste comenzi:

```
% mysql -h localhost test
% mysql -h pit-viper.snake.net test
```

Faptul că vă puteți conecta la serverul MySQL ca utilizator `root` fără nici o parolă înseamnă că instalarea inițială nu este sigură. Ca atare, una dintre primele dumneavoastră operații în calitate de administrator al sistemului MySQL trebuie să fie configurarea parolei pentru utilizatorul `root`. Apoi, în funcție de metoda pe care o folosiți pentru a configura parola, poate fi necesar să comunicați serverului să reîncarce tabelele de acordare, astfel încât să sesizeze modificarea. (Când serverul pornește, încarcă tabelele în memorie și s-ar putea să nu recunoască modificările efectuate. În acest caz, trebuie să indicați în mod explicit serverului să recitească tabelele.)

Pentru MySQL versiunea 3.22 și versiunile ulterioare, puteți stabili parola folosind programul `mysqladmin` așa cum se va vedea mai jos, înlocuind expresia `"parola mea"` din comandă cu parola pe care ați ales-o:

```
% mysqladmin -u root password "parola mea"
```

Pentru orice versiune de MySQL, puteți folosi programul `mysql` și puteți actualiza direct tabelul de acordare `user` din baza de date `mysql`:

```
% mysql -u root mysql
mysql> UPDATE user SET Password=PASSWORD("parola mea")
    -> WHERE User="root";
```

Dacă aveți o versiune mai veche de MySQL, folosiți `mysql` și o instrucțiune `UPDATE`; procedeul este variabil și pentru versiunea shareware sub Windows.

După ce ați configurat parola, rulați următoarea comandă, pentru a afla dacă trebuie să indicați serverului să reîncarce tabelele de acordare:

```
% mysqladmin -u root status
```

Dacă serverul vă mai permite să vă conectați ca `root` fără o parolă, indicați-i să recitească tabelele de acordare astfel:[1]

```
% mysqladmin -u root reload
```

[1] Începând de la versiunea 3.22.11, MySQL vă permite să reîncărcați tabelele folosind și comanda `mysqladmin flush-privileges`, respectiv instrucțiunea SQL `FLUSH PRIVILEGES`. – N.A.

După ce ați configurat parola utilizatorului `root` (și ați reîncărcat tabelele de acordare, dacă este necesar), va trebui să specificați noua dumneavoastră parolă de fiecare dată când vă veți conecta la server sub contul `root`.

Configurarea pornirii și opririi serverului MySQL

Ca administrator al serverului MySQL, una dintre atribuțiile dumneavoastră generale este aceea de a vă asigura că serverul rulează cât mai mult timp posibil, astfel încât clienții să aibă acces la acesta. Uneori, însă, cel mai bine este să opriți serverul. (De exemplu, dacă modificați amplasamentul unei baze de date, nu doriți ca serverul să actualizeze tabelele din baza de date respectivă.) Tensiunea creată între necesitatea de a păstra serverul în funcțiune și obligația de a-l opri ocazional nu este un subiect care poate fi rezolvat în această carte. Cel puțin, însă, putem discuta despre modul de pornire și oprire a serverului astfel încât dumneavoastră să aveți posibilitatea de a executa oricare dintre aceste operații oricând credeți de cuviință.

Instrucțiunile din acest capitol se aplică numai sistemelor UNIX. Dacă rulați sub Windows, le puteți ignora, deoarece în Anexa A, „Obținerea și instalarea programelor", se găsesc toate comenzile de pornire și de oprire de care aveți nevoie.

Invocarea comenzilor prezentate în acest capitol

În majoritatea cazurilor, programe precum `mysqladmin`, `mysqldump` și altele sunt prezentate în acest capitol fără nici una din opțiunile `-h`, `-u` sau `-p`, din motive de simplitate. Voi presupune că veți invoca programele cu toate opțiunile necesare pentru a stabili o conexiune cu serverul.

Rularea serverului MySQL folosind un cont de utilizator fără privilegii

Înainte de a discuta despre modul de pornire a serverului, să vedem care este contul sub care serverul ar trebui să ruleze la pornire. Serverul poate fi pornit manual sau automat. Dacă îl porniți manual, serverul rulează sub contul utilizatorului UNIX în cadrul căruia ați deschis sesiunea de lucru. Cu alte cuvinte, dacă deschid sesiunea de lucru sub numele `paul` și pornesc serverul, acesta va rula sub contul `paul`. Dacă, în schimb, comut utilizatorul la `root` folosind comanda `su` și apoi pornesc serverul, acesta va rula sub contul `root`.

Cu toate acestea, în majoritatea cazurilor nu veți dori să porniți manual serverul. Cel mai probabil, veți configura serverul astfel încât să pornească automat, la încărcarea sistemului de operare, ca parte a procedurii standard de pornire. Sub UNIX, această procedură de pornire este executată de sistem sub contul utilizatorului `root` din UNIX, iar toate procesele pornite pe durata procedurii vor rula cu privilegii `root`.

Trebuie să rețineți două scopuri ale procedurilor dumneavoastră de pornire a serverului MySQL:

- **Doriți ca serverul să ruleze sub un cont de utilizator altul decât `root`.** În general, doriți să limitați puterea oricărui proces, cu excepția situațiilor când are nevoie de acces de tip `root`, iar `mysqld` nu are nevoie de un asemenea acces.
- **Doriți ca utilizatorul să ruleze mereu sub același cont de utilizator.** Este lipsit de consecvență ca serverul să ruleze uneori sub un cont de utilizator, iar alteori sub un alt cont de utilizator. Acest lucru duce la crearea de fișiere și cataloage cu date cu proprietari diferiți și pot avea ca rezultat chiar incapacitatea serverului de a obține accesul la bazele de date sau la tabele, în funcție de contul de utilizator sub care rulează. Rularea în mod consecvent a serverului sub același cont de utilizator vă permite să evitați această problemă.

Pentru a rula serverul sub un cont de utilizator normal, fără privilegii, parcurgeți următoarea procedură:

1. Alegeți un cont pentru a-l utiliza la rularea serverului. `mysqld` poate rula sub orice cont de utilizator, dar, din punct de vedere conceptual, este mai simplu să creați un cont separat pentru activitatea în sistemul MySQL. De asemenea, puteți desemna un grup special pentru MySQL. Voi denumi aceste nume de utilizator și de grup `mysqladm`, respectiv `mysqlgrp`. Dacă folosiți alte nume, înlocuiți toate mențiunile `mysqladm` și `mysqlgrp` existente pe parcursul acestei cărți cu numele respective.

   Dacă ați instalat MySQL sub propriul dumneavoastră cont și nu aveți privilegii administrative speciale în sistemul dumneavoastră, probabil că veți rula serverul sub propriul dumneavoastră identificator de utilizator. În acest caz, înlocuiți `mysqladm` și `mysqlgrp` cu numele dumneavoastră de deschidere a sesiunii de lucru, respectiv numele de grup.

   Dacă ați instalat MySQL sub RedHat Linux folosind un fișier RPM, la instalare s-a creat în mod automat un cont sub numele `mysql`. Înlocuiți `mysqladm` cu numele respectiv.
2. Creați contul de server, dacă este necesar, folosind procedura uzuală pentru crearea conturilor din sistemul dumneavoastră. Această operație trebuie executată din contul `root`.
3. Opriți serverul, dacă acesta funcționează.
4. Modificați proprietarul catalogului de date și al tuturor sub-cataloagelor și fișierelor, astfel încât utilizatorul `mysqladm` să fie posesorul acestora. De exemplu, în cazul în care catalogul de date este `/usr/local/var`, puteți stabili proprietarul catalogului `mysqladm` după cum urmează (aceste comenzi trebuie executate din contul `root`):

`# cd /usr/local/var`	Mutare în catalogul de date
`# chown -R mysqladm.mysqlgrp`	Stabilește proprietarul tuturor cataloagelor și al fișierelor

5. Modificați permisiunile aferente catalogului de date, precum și cele aferente tuturor cataloagelor și fișierelor, astfel încât numai utilizatorul `mysqladm` să poată avea acces la ele. Configurarea modului astfel încât accesul altor persoane să fie interzis este o bună măsură de securitate. În cazul în care catalogul de date este `/usr/local/var`, puteți configura toate obiectele ca fiind în posesia utilizatorului `mysqladm`, după cum urmează (aceste comenzi trebuie executate din contul `root`):

`# cd /usr/local/var`	Mutare în catalogul de date
`# chmod -R go-rwx`	Pune toate obiectele numai la dispoziția utilizatorului `mysqladm`

Când configurați proprietarul și modul catalogului de date și al conținutului acestuia, fiți atent la legăturile simbolice. Va trebui să le urmați și să modificați proprietarul și modul fișierelor sau al cataloagelor spre care indică acestea. Această operație poate fi problematică în cazul în care catalogul în care se găsesc fișierele legate nu vă aparține, deci poate fi necesar să lucrați sub contul `root`.

După ce ați finalizat procedura anterioară, trebuie întotdeauna să porniți serverul fie sub contul `mysqladm`, fie sub contul `root`. În ultimul caz, nu uitați să specificați opțiunea `--user=mysqladm`, astfel încât utilizatorul să poată înlocui identificatorul său de utilizator cu acela al utilizatorului `mysqladm` (fapt de asemenea valabil și în cazul procedurii de pornire a sistemului).

Opțiunea `--user` a fost adăugată la `mysql` în MySQL versiunea 3.22. Dacă aveți o versiune mai veche, puteți folosi comanda su pentru a cere sistemului să ruleze serverul sub un anumit cont, atunci când porniți serverul în timp ce rulați ca `root`. Va trebui să citiți pagina de manual a comenzii su, deoarece sintaxa de rulare a unei comenzi ca utilizator dat variază.

Metode pentru pornirea serverului

După ce v-ați decis cu privire la contul pe care îl veți folosi pentru rularea serverului, puteți alege configurarea modului de pornire a serverului. Puteți rula serverul manual (de la linia de comandă), respectiv automat (în timpul procedurii de pornire a sistemului). Există trei metode principale pentru pornirea serverului:

- **Invocarea directă a programului** `mysqld`. Aceasta este, probabil, metoda cea mai rar folosită. Nu voi spune mai multe despre această metodă, cu excepția faptului că `mysqld --help` este o comandă utilă pentru a descoperi opțiunile pe care le puteți folosi cu celelalte metode de pornire.
- **Invocarea scriptului** `safe_mysqld`. Scriptul `safe_mysqld` încearcă să determine locația programului server și a catalogului de date, după care invocă serverul cu opțiuni care reflectă aceste valori. Scriptul `safe_mysqld` modifică direcția ieșirii de eroare standard de la server la un fișier de eroare amplasat în catalogul de date, astfel încât să existe o înregistrare a acesteia. După ce pornește serverul, `safe_mysqld` îl monitorizează și îl repornește în caz că serverul își încheie execuția. Scriptul `safe_mysqld` este frecvent folosit în versiunile UNIX de tip BSD.

  Jurnalul de eroare se va afla în posesia utilizatorului `root`, dacă porniți vreodată scriptul `safe_mysqld` sub contul `root` sau în timpul pornirii sistemului. Acest lucru poate cauza mai târziu apariția unei erori de tip „permisiune refuzată“, dacă încercați să invocați scriptul `safe_mysqld` ca utilizator fără privilegii. Eliminați jurnalul de eroare și încercați din nou.
- **Invocarea scriptului** `mysql.server`. Acest script pornește serverul rulând `safe_mysqld`. `mysql.server` este destinat a fi utilizat în sisteme care folosesc sistemul de pornire-oprire System V. Acest sistem include numeroase cataloage care conțin scripturile ce urmează a fi invocate atunci când calculatorul intră sau părăsește un nivel de rulare dat. Scripturile pot fi invocate cu argumentul `start` sau `stop`, pentru a arăta dacă doriți ca serverul să pornească sau să se oprească.

Scriptul `safe_mysqld` este instalat în catalogul `bin`, situat în catalogul de instalare MySQL, sau poate fi găsit în catalogul `scripts` al distribuției sursă MySQL. Scriptul `mysql.server` este instalat în catalogul `share/mysql` din catalogul de instalare al sistemului MySQL, sau se poate găsi în catalogul `support-files` al distribuției sursă MySQL. Dacă doriți să folosiți acest script, trebuie să-l copiați în catalogul de pornire adecvat.

Pentru sisteme de tip BSD, se obișnuiește să se insereze în catalogul `/etc` cataloage care inițiază serviciile la momentul inițializării sistemului de operare. Aceste fișiere au deseori nume care încep cu `rc` și este posibil să existe un fișier cu numele `rc.local` (sau ceva similar) special conceput pentru pornirea serviciilor instalate local. Pe un asemenea sistem, puteți adăuga la fișierul `rc.local` linii precum cele prezentate mai jos pentru a porni serverul (modificați calea către `safe_mysqld`, dacă aceasta este diferită în sistemul dumneavoastră):

```
if [ -x /usr/local/bin/safe_mysqld ]; then
    /usr/local/bin/safe_mysqld &
fi
```

Pentru sistemele de tip System V, puteți instala scriptul `mysql.server` prin amplasarea sa în catalogul de pornire adecvat din catalogul `/etc`. Dacă rulați Linux și ați instalat MySQL dintr-un fișier RPM, această operație a fost efectuată automat. În caz contrar, instalați scriptul în catalogul principal cu scripturi de pornire și plasați legături spre acesta în catalogul de rulare de nivel adecvat. De asemenea, puteți face scriptul executabil numai din contul `root`.

Structura cataloagelor cu fișiere de pornire variază de la un sistem la altul, deci veți avea nevoie să examinați modul în care le organizează sistemul dumneavoastră. De exemplu, în LinuxPPC cataloagele sunt `/etc/rc.d/init.d` și `/etc/rc.d/rc3.d`. Un script se poate instala astfel:

```
# cp mysql.server /etc/rc.d/init.d
# cd /etc/rc.d/init.d
# chmod 500 mysql.server
# cd /etc/rc.d/rc3.d
# ln -s ..init.d/mysql.server S99mysql
```

În Solaris, catalogul principal cu scripturi este `/etc/init.d`, iar catalogul nivelului de rulare este `/etc/rc2.d`, deci comenzile vor avea următorul aspect:

```
# cp mysql.server /etc/init.d
# cd /etc/init.d
# chmod 500 mysql.server
# cd /etc/rc2.d
# ln -s ..init.d/mysql.server S99mysql
```

În momentul pornirii sistemului, scriptul `S99mysql` va fi invocat automat, cu argumentul `start`.

Dacă aveți o comandă `chkconfig` (folosită frecvent sub Linux, de exemplu), o puteți utiliza pentru a vă ajuta să instalați scriptul `mysql.server`, în loc de a rula manual comenzi precum cele prezentate anterior.

Specificarea opțiunilor de pornire

Dacă doriți să specificați opțiuni de pornire suplimentare în momentul pornirii serverului, puteți proceda în două moduri. Puteți modifica scriptul de pornire pe care îl folosiți (`safe_mysqld` sau `mysql.server`) și specificați opțiunile direct în linia care invocă serverul. Alternativ, puteți specifica opțiunile într-un fișier cu opțiuni. Vă recomand să specificați opțiunile pentru server într-un fișier cu opțiuni global, dacă este posibil. Amplasarea uzuală a acestuia este `/etc/my.cnf` în UNIX, respectiv `C:\my.cnf` sub Windows. (Pentru detalii privind utilizarea fișierelor cu opțiuni, consultați Anexa E.)

Unele categorii de informații nu pot fi specificate ca opțiuni ale serverului. Pentru acestea, va trebui probabil să modificați scriptul `safe_mysqld`. De exemplu, dacă serverul dumneavoastră nu alege în mod adecvat fusul orar local și returnează valori ale orelor exprimate în ore GMT, puteți configura variabila de mediu `TZ` pentru a-i da o sugestie. Dacă porniți serverul folosind `safe_mysqld` sau `mysql.server`, puteți adăuga la `safe_mysqld` un parametru legat de fusul orar. Descoperiți linia care pornește serverul și adăugați înaintea liniei respective o comandă ca aceasta:

```
TZ=US/Central
export TZ
```

Astfel, `TZ` este configurată la fusul orar US Central (ora civilă corespunzătoare meridianului 105 – N.T.) Va trebui să folosiți fusul orar adecvat pentru amplasamentul dumneavoastră. Sintaxa prezentată se utilizează în sistemele Solaris; sistemul dumneavoastră poate folosi o sintaxă diferită. De exemplu, o altă sintaxă comună pentru configurarea variabilei `TZ` este următoarea:

```
TZ=CST6CDT
export TZ
```

Dacă vă modificați scriptul de pornire, rețineți că data viitoare când veți instala MySQL (de exemplu pentru a trece la o versiune mai recentă), modificările dumneavoastră se vor pierde dacă nu copiați în prealabil scriptul de pornire într-o altă locație. După instalarea noii versiuni, comparați scriptul dumneavoastră cu scriptul nou instalat, pentru a vedea care sunt modificările pe care trebuie să le efectuați din nou.

Verificarea tabelelor la pornire

În afară de a vă configura serverul astfel încât să pornească la încărcarea sistemului de operare, puteți dori să instalați un script care să ruleze `myisamchk` și `isamchk` pentru a vă verifica tabelele înainte de pornirea serverului. Poate că reporniți serverul în urma unei căderi și este posibil ca tabelele să se fi deteriorat. Verificarea tabelelor înainte de pornirea serverului constituie o bună modalitate de a descoperi eventualele probleme. Capitolul 13 conține detalii cu privire la scrierea și instalarea unui asemenea script.

Oprirea serverului

Pentru a opri manual serverul, folosiți `mysqladmin`:

```
% mysqladmin shutdown
```


Pentru a închide automat serverul, nu trebuie să executați nici o operație specială. În mod normal, sistemele BSD opresc serviciile trimițând proceselor un semnal `TERM`. Procesele fie răspund în mod adecvat la acest semnal, fie sunt suprimate fără prea multe discuții. La primirea acestui semnal, `mysqld` răspunde prin terminarea execuției. Pentru sistemele de tip System V, care pornesc serverul cu `mysql.server`, procesul de oprire va invoca scriptul respectiv cu argumentul `stop`, pentru a indica serverului să-și încheie execuția – presupunând că ați instalat `mysql.server`, desigur...

Redobândirea controlului asupra serverului în cazul imposibilității de conectare

În anumite situații, trebuie să reporniți manual serverul, datorită incapacității de a vă conecta la acesta. Desigur, situația este oarecum paradoxală, deoarece oprirea manuală a serverului se execută, în mod caracteristic, prin conectarea la server și emiterea comenzii de sistare a execuției. Atunci, cum poate apărea o asemenea situație?

Mai întâi, este posibil ca parola `root` să fi primit o valoare pe care dumneavoastră nu o cunoașteți. Acest lucru se poate întâmpla când modificați parola – de exemplu, dacă introduceți în mod accidental un caracter invizibil de control atunci când introduceți noua valoare a parolei. De asemenea, s-ar putea ca pur și simplu să fi uitat parola.

În al doilea rând, conexiunile la `localhost` se stabilesc, de regulă, prin intermediul unui fișier soclu de domeniu UNIX, care de obicei este `/tmp/mysql.sock`. Dacă fișierul soclu este eliminat, clienții locali nu se vor mai putea conecta. Acest lucru se poate întâmpla dacă sistemul dumneavoastră rulează o sarcină `cron` care elimină ocazional fișierele temporare din catalogul `/tmp`.

Dacă nu vă puteți conecta deoarece ați pierdut fișierul soclu, îl puteți obține prin simpla repornire a serverului, deoarece serverul îl creează din nou la pornire. Poanta este că nu puteți folosi soclul pentru a stabili o conexiune, fiindcă s-a pierdut. În schimb, trebuie să stabiliți o conexiune TCP/IP. De exemplu, dacă gazda serverului este `pit-viper.snake.net`, vă puteți conecta astfel:

```
% mysqladmin -p -u root -h pit-viper.snake.net shutdown
```

Dacă fișierul soclu este eliminat de o sarcină `cron`, problema va persista până la modificarea sarcinii `cron` sau până la utilizarea unui alt fișier soclu. Puteți specifica un alt soclu folosind fișierul global cu opțiuni. De exemplu, în cazul în care catalogul de date este `/usr/local/var`, puteți deplasa acolo fișierul soclu adăugând următoarele linii la fișierul `/etc/my.cnf`:

```
[mysqld]
socket=/usr/local/var/mysql.sock

[client]
socket=/usr/local/var/mysql.sock
```

Calea de acces este specificată atât pentru server, cât și pentru programele client, astfel încât toate să folosească același fișier soclu. În cazul în care configurați calea de acces numai pentru server, programele client vor continua să presupună că soclul se află la locația veche. Reporniți serverul după ce ați operat modificarea, astfel încât acesta să creeze soclul în noua locație.

Dacă nu vă puteți conecta deoarece nu vă puteți aminti parola contului root sau i-ați atribuit o valoare care diferă de ceea ce știați dumneavoastră, trebuie să recâștigați controlul asupra serverului, astfel încât să puteți configura din nou parola:

- **Opriți serverul.** Dacă puteți deschide sesiunea de lucru ca root pe gazda serverului, puteți termina serverul folosind comanda `kill`. Puteți afla identificatorul de proces al serverului folosind comanda `ps` sau examinând fișierul PID al serverului (care se află, de obicei, în catalogul de date).

  Cel mai bine este să încercați mai întâi să suprimați serverul cu o instrucțiune `kill` normală, care trimite serverului un semnal `TERM`, pentru a vedea dacă serverul răspunde printr-o oprire normală. Astfel, tabelele și jurnalele vor fi golite de conținut în mod adecvat. Dacă serverul este blocat și nu reacționează la un semnal de terminare normal, veți folosi instrucțiunea `kill -9` pentru a-l termina cu forța. Aceasta constituie numai o ultimă soluție, deoarece pot exista modificări care nu au fost eliminate, iar dumneavoastră riscați să lăsați tabelele într-o stare inconsecventă.

  Dacă ați terminat serverul cu instrucțiunea `kill -9`, nu uitați să verificați tabelele cu `isamchk` și `myisamchk` înainte de a reporni serverul (vezi capitolul 13).
- **Reporniți serverul cu opțiunea `--skip-grant-tables`.** Această opțiune indică serverului să nu folosească tabelele de acordare pentru verificarea conexiunilor. Astfel, veți putea să vă conectați ca `root`, fără nici o parolă. După ce v-ați conectat, modificați parola `root` așa cum s-a arătat în secțiunea „Asigurarea unei instalări MySQL recente".
- **Indicați serverului să înceapă a folosi din nou tabelele de acordare, cu ajutorul opțiunii `mysqladmin flush-privileges`.** Dacă versiunea programului `mysqladmin` pe care o dețineți nu recunoaște opțiunea `flush-privileges`, încercați în schimb comanda `reload`.

Gestionarea conturilor de utilizator

Administratorul MySQL trebuie să știe cum să configureze conturile de utilizator MySQL, specificând utilizatorii care se pot conecta la server, sistemele de unde se pot conecta și acțiunile care le sunt permise în timpul conexiunii. MySQL 3.22.11 a introdus două instrucțiuni care simplifică în mare măsură această sarcină: instrucțiunea `GRANT` creează utilizatori MySQL și le specifică privilegiile, iar instrucțiunea `REVOKE` elimină privilegiile. Aceste două instrucțiuni se comportă ca o interfață în raport cu tabelele de acordare din baza de date `mysql` și furnizează o alternativă la manipularea directă a conținutului acestor tabele. Instrucțiunile `GRANT` și `REVOKE` afectează patru tabele:

Tabel de acordare	Conținut
`user`	Utilizatorii care se pot conecta la server și toate privilegiile globale pe care le au aceștia
`db`	Privilegii la nivel de bază de date
`tables_priv`	Privilegii la nivel de tabel
`columns_priv`	Privilegii la nivel de coloană

Există și un al cincilea tabel de acordare, (`host`), dar acesta nu este afectat de cele două instrucțiuni.

Când emiteți o instrucțiune `GRANT` pentru un utilizator, este creată o intrare pentru utilizatorul respectiv în tabelul `user`. Dacă instrucțiunea specifică privilegii globale (privilegii administrative sau care se aplică tuturor bazelor de date), acestea sunt de asemenea înregistrate în tabelul `user`. Dacă specificați privilegii la nivel de bază de date, tabel sau coloană, acestea sunt înregistrate respectiv în tabelele `db`, `tables_priv` și `columns_priv`.

A folosi instrucțiunile `GRANT` și `REVOKE` este mai simplu decât a modifica în mod direct tabelele de acordare. Totuși, este bine să completați materialul din capitolul de față citind capitolul 12, care discută despre tabelele de acordare într-o manieră mai detaliată. Aceste tabele sunt extrem de importante și un administrator trebuie să înțeleagă modul lor de funcționare dincolo de nivelul instrucțiunilor `GRANT` și `REVOKE`.

În continuarea acestei secțiuni, vom descrie modul de configurare a conturilor de utilizator MySQL și modul de acordare a privilegiilor. De asemenea, vom discuta despre modalitatea de revocare a privilegiilor și vom aborda o „enigmă a privilegiilor" care ridică probleme multor administratori MySQL începători.

De asemenea, veți dori să luați în considerare și utilizarea scripturilor `mysqlaccess` și `mysql_setpermission`, care fac parte din distribuția MySQL. Acestea sunt scripturi Perl care constituie o alternativă față de instrucțiunea `GRANT` pentru configurarea conturilor de utilizator. Scriptul `mysql_setpermission` impune instalarea suportului pentru DBI.

Crearea noilor utilizatori și acordarea privilegiilor

Sintaxa instrucțiunii `GRANT` este următoarea:

```
GRANT privilegii [coloane]
    ON entitate
    TO utilizator IDENTIFIED BY "parola"
    WITH GRANT OPTION
```

Pentru a folosi instrucțiunea, trebuie să completați următoarele părți:

- *`privilegii`*

  Acestea sunt privilegiile care vor fi atribuite utilizatorului. Specificatorii de privilegii care pot fi folosiți cu instrucțiunea `GRANT` sunt prezentați în tabelul următor:

Specificator de privilegiu	Operația permisă de privilegiu
`ALTER`	Modifică tabelele și indexurile
`CREATE`	Creează baze de date și tabele
`DELETE`	Șterge din tabele înregistrările existente
`DROP`	Elimină bazele de date și tabelele
`INDEX`	Creează sau șterge indexuri
`INSERT`	Inserează în tabele înregistrări noi
`REFERENCES`	Nefolosit
`SELECT`	Regăsește din tabele înregistrările existente

UPDATE	Modifică înregistrările existente în tabele
FILE	Citește și scrie fișierele la server
PROCESS	Vizualizează informații despre firele aflate în execuție la server sau suprimă firele
RELOAD	Reîncarcă tabelele de acordare sau golește jurnalele, memoria cache a gazdei sau memoria cache cu tabele
SHUTDOWN	Oprește serverul
ALL	Toate operațiile; sinonim cu ALL PRIVILEGES
USAGE	Un privilegiu special „fără privilegii"

Specificatorii de privilegii din primul grup prezentat în tabelul precedent se aplică bazelor de date, tabelelor și coloanelor. Specificatorii din al doilea grup reprezintă privilegii administrative. În mod normal, aceste privilegii sunt acordate cu economie, deoarece permit utilizatorilor să afecteze modul de operare a serverului. (De exemplu, privilegiul SHUTDOWN nu este unul pe care să-l acordați zilnic.) Specificatorii din al treilea grup sunt speciali. ALL înseamnă „toate privilegiile". USAGE înseamnă „fără privilegii" – cu alte cuvinte creează utilizatorul, dar nu-i acordă nici un fel de privilegii.

- *coloane*

  Coloanele cărora li se aplică privilegiile. Acest parametru este opțional, iar dumneavoastră îl folosiți numai pentru a configura privilegii specifice coloanelor. Dacă sunt specificate minimum două coloane, acestea trebuie separate prin virgulă.

- *entitate*

  Nivelul la care se aplică privilegiile. Privilegiile pot fi globale (se aplică tuturor bazelor de date și tuturor tabelelor), specifice bazelor de date (se aplică tuturor tabelelor dintr-o bază de date) sau specifice tabelelor. Privilegiile pot deveni specifice coloanelor prin specificarea unei clauze coloane.

- *utilizator*

  Utilizatorul căruia îi sunt destinate privilegiile. Parametrul este alcătuit dintr-un nume de utilizator și un nume de gazdă; în MySQL, specificați nu numai *cine* se poate conecta, dar și *de unde* se poate conecta. Astfel, puteți avea doi utilizatori cu același nume, care se conectează de la locații diferite. MySQL vă permite să faceți diferența între ei, precum și să acordați fiecăruia privilegii în mod independent de celălalt.

  În MySQL, un nume de utilizator este doar numele pe care îl specificați atunci când vă conectați la server. Numele nu trebuie să aibă nici o legătură cu numele dumneavoastră UNIX de deschidere a sesiunii de lucru și nici cu numele Windows. În mod prestabilit, programele client vor folosi numele dumneavoastră de deschidere a sesiunii de lucru ca nume de utilizator MySQL, dacă nu specificați un nume în mod explicit, dar aceasta nu este decât o convenție. De asemenea, numele root acordat super-utilizatorului din MySQL care poate executa orice operație nu are nimic special. Nu este decât o convenție. Puteți înlocui acest nume cu numele nimeni în tabelele de acordare, după care vă conectați ca nimeni pentru a executa operații care necesită privilegii de super-utilizator.

- *parola*

  Parola care va fi atribuită utilizatorului. Este un parametru opțional. Dacă nu specificați o clauză IDENTIFIED BY pentru un utilizator nou, acel utilizator nu va primi nici o parolă (ceea ce este contraindicat din punct de vedere al siguranței). Pentru un utilizator existent, orice parolă pe care o specificați o înlocuiește pe cea veche. Dacă nu specificați o parolă nouă, vechea parolă a utilizatorului rămâne neschimbată. Când folosiți clauza IDENTIFIED BY, șirul parolei trebuie să fie textul literal al parolei; instrucțiunea GRANT va codifica automat parola. Nu folosiți funcția PASSWORD() așa cum procedați în cazul instrucțiunii SET PASSWORD.

Clauza WITH GRANT OPTION este opțională. Dacă o includeți, utilizatorul poate atribui altor utilizatori toate privilegiile pe care le poate acorda instrucțiunea GRANT. Puteți folosi această clauză pentru a delega altor utilizatori funcționalitățile de acordare a privilegiilor.

În cadrul intrărilor din tabelele de acordare, numele de utilizatori, parolele, precum și numele bazelor de date și ale tabelelor sunt sensibile la diferența între majuscule și minuscule. Numele gazdelor și numele coloanelor nu prezintă această sensibilitate.

În general, puteți determina categoria de instrucțiune GRANT de care aveți nevoie punând unele întrebări:

- Cine se poate conecta și de unde?
- Ce nivel de privilegii trebuie să aibă utilizatorul și cărei entități trebuie să se aplice acestea?
- Trebuie ca utilizatorul să aibă permisiunea de a administra privilegiile?

Să punem aceste întrebări și să examinăm unele exemple de utilizare a instrucțiunii GRANT pentru configurarea conturilor de utilizator MySQL.

Cine se poate conecta și de unde?

Puteți permite unui utilizator să se conecteze de la un set de gazde a cărui extindere rămâne la latitudinea dumneavoastră. La o extremă, puteți limita accesul la o singură gazdă, dacă știți că utilizatorii se vor conecta numai de la gazda respectivă:

```
GRANT ALL ON samp_db.* TO boris@localhost IDENTIFIED BY "rubin"
GRANT ALL ON samp_db.* TO fred@ares.mars.net IDENTIFIED BY "cuart"
```

(Notația samp_db.* are semnificația „toate tabelele din baza de date samp_db".) La cealaltă extremă, puteți avea un utilizator max care călătorește foarte mult și trebuie să se poată conecta de la gazde situate oriunde în lume. În acest caz, îi puteți permite să se conecteze, indiferent de unde se conectează:

```
GRANT ALL ON samp_db.* TO max@% IDENTIFIED BY "diamant"
```

Caracterul % funcționează drept caracter de înlocuire, cu aceeași semnificație ca la stabilirea unei corespondențe cu un model, unde se folosește clauza LIKE. În instrucțiunea precedentă, are semnificația „orice gazdă". Dacă nu furnizați nici un nume al gazdei, este același lucru cu a specifica opțiunea %. Astfel, notațiile max și max@% sunt echivalente. Aceasta este modalitatea cea mai simplă, dar și cea mai puțin sigură de configurare a unui utilizator.

Pentru a aborda o cale de mijloc, puteți permite unui utilizator să se conecteze de la un set limitat de gazde. De exemplu, pentru a permite lui `mary` să se conecteze de la orice gazdă din cadrul domeniului `snake.net`, folosiți un specificator al numelui gazdei `%.snake.net`:

```
GRANT ALL ON samp_db.* TO mary@%.snake.net IDENTIFIED BY "topaz"
```

Partea cu numele gazdei din identificatorul de utilizator se poate preciza folosind o adresă IP în locul numelui gazdei, dacă doriți. Puteți specifica o adresă IP literală sau o adresă care conține caractere de model. De asemenea, începând de la MySQL 3.23, puteți specifica numere IP cu o mască de rețea, care să indice numărul de biți folosiți pentru numărul rețelei:

```
GRANT ALL ON samp_db.* TO joe@192.168.128.3 IDENTIFIED BY "apa"
GRANT ALL ON samp_db.* TO ardis@192.168.128.% IDENTIFIED BY "zapada"
GRANT ALL ON samp_db.* TO rex@192.168.128.0/17 IDENTIFIED BY "gheata"
```

Prima din aceste instrucțiuni indică o anumită gazdă de la care se poate conecta utilizatorul. Cea de-a doua specifică un model IP pentru subrețeaua de clasă C 192.168.128. În cea de-a treia instrucțiune, `192.168.128.0/17` specifică un număr de rețea pe 17 biți și corespunde oricărei gazde care conține valoarea `192.168.128` în primii 17 biți ai adresei sale IP.

Dacă MySQL face reclamații cu privire la valoarea utilizator pe care o specificați, poate fi necesar să utilizați ghilimele (dar încadrați separat între ghilimele părțile cu numele de utilizator și numele gazdei):

```
GRANT ALL ON samp_db.presedinte TO "amicul meu"@"boa.snake.net"
```

Ce nivel de privilegii trebuie să aibă utilizatorul și cărei entități trebuie să se aplice acestea?

Puteți acorda diferite niveluri de privilegii. Privilegiile globale sunt cele mai puternice, deoarece se aplică oricărei baze de date. Pentru a face din `elena` un super-utilizator care poate executa orice operație, având inclusiv capacitatea de a acorda privilegii altor utilizatori, emiteți următoarea instrucțiune:

```
GRANT ALL ON *.* elena@localhost IDENTIFIED BY "cafea"
   WITH GRANT OPTION
```

Specificatorul `*.*` din clauza `ON` are semnificația „toate bazele de date și toate tabelele". Ca măsură de precauție, am precizat că `elena` se poate conecta numai de la gazda locală. Limitarea gazdelor de la care se poate conecta un super-utilizator este, de regulă, o operație inteligentă, deoarece limitează gazdele de la care se pot iniția încercări de „spargere" a parolelor.

Unele privilegii (`FILE`, `PROCESS`, `RELOAD` și `SHUTDOWN`) sunt administrative și pot fi acordate folosind numai specificatorul de privilegiu global `ON *.*`. Dacă doriți, puteți acorda aceste privilegii fără a mai atribui și privilegii la nivel de bază de date. De exemplu, următoarea instrucțiune configurează un utilizator `flush` care nu poate decât să emită instrucțiuni `FLUSH`. Acest lucru poate fi util în scripturile administrative în care aveți de efectuat operații cum ar fi golirea jurnalelor în timpul rotației fișierelor jurnal:

```
GRANT RELOAD ON *.* TO flush@localhost IDENTIFIED BY "flushpass"
```


În general, este de dorit ca acordarea privilegiilor administrative să se execute cu economie, deoarece utilizatorii care le dețin pot afecta modul de operare a serverului dumneavoastră.

Privilegiile la nivel de bază de date se aplică tuturor tabelelor dintr-o anumită bază de date. Aceste privilegii se acordă folosind o clauză `ON nume_baza_de_date.*`:

```
GRANT ALL ON samp_db.* TO bill@racer.snake.net IDENTIFIED BY "piatra"
GRANT SELECT ON menajerie.* TO nc_user@% IDENTIFIED BY "noroi"
```

Prima dintre aceste instrucțiuni acordă utilizatorului `bill` privilegii complete pentru orice tabel din baza de date `samp_db`. Cea de-a doua creează un utilizator cu acces restrâns `nc_user` („utilizator numai pentru citire") care poate obține accesul la orice tabel din baza de date `menajerie`, dar numai pentru citire. Cu alte cuvinte, utilizatorul poate emite numai instrucțiuni `SELECT`.

Modul de specificare a numelui gazdei locale în intrările din tabelele de acordare

Este normal să aveți probleme la conectarea de la gazda serverului dacă folosiți numele gazdei serverului și nu `localhost`. Această situație poate apărea datorită unei nepotriviri între modul de specificare a numelui în tabelele de acordare și modalitatea în care rutinele dumneavoastră de conversie a numelui raportează aceste nume programelor. Dacă rutina de conversie raportează un nume incomplet determinat, precum `pit-viper`, dar tabelele de acordare conțin intrări cu numele complet - cum ar fi `pit-viper.snake.net` (sau invers) - se va produce această neconcordanță.

Pentru a determina dacă există asemenea situații în sistemul dumneavoastră, încercați să vă conectați la serverul local folosind o opțiune `-h`, care specifică numele gazdei dumneavoastră. Apoi, examinați fișierul jurnal general al serverului. Cum raportează acesta numele gazdei? Este acesta în formă complet determinată sau nu? Indiferent de forma în care apare, vă indică modul în care trebuie să precizați partea cu numele gazdei din specificatorii de utilizator, atunci când emiteți instrucțiuni `GRANT`.

Puteți menționa un număr de privilegii individuale care să fie acordate simultan. De exemplu, dacă doriți ca un utilizator să poată citi și opera modificări în conținutul tabelelor existente, dar fără a putea să creeze tabele noi sau să șteargă tabele, acordați aceste privilegii după cum urmează:

```
GRANT SELECT,INSERT,DELETE,UPDATE ON samp_db.* TO jennie@%
   IDENTIFIED BY "bor"
```

Pentru un control mai strict al accesului, puteți acorda privilegii asupra tabelelor individuale și chiar asupra coloanelor individuale din tabele. Privilegiile specifice coloanelor sunt utile atunci când există părți ale unui tabel pe care doriți să le ascundeți de un anumit utilizator, respectiv când doriți ca utilizatorul să poată modifica numai anumite coloane. Să presupunem că, la secretariatul Ligii istorice, o persoană se oferă să vă ajute în îndeplinirea obligațiilor dumneavoastră ca secretar al Ligii. Asta-i o veste bună, dar vă decideți să începeți prin a acorda noului dumneavoastră asistent acces numai pentru citire la tabelul `membru` care conține informațiile despre membri, plus o un privilegiu de coloană `UPDATE` asupra coloanei `data_expirare` din tabelul respectiv. În acest mod, asistentul dumneavoastră poate începe cu modesta sarcină de a actualiza datele de expirare, pe măsură ce membrii își plătesc cotizațiile. Instrucțiunile pentru configurarea acestui utilizator MySQL sunt după cum urmează:

```
GRANT SELECT ON samp_db.membru
   TO asistent@localhost IDENTIFIED BY "ajutorlabirou"
```

```
GRANT UPDATE (data_expirare) ON samp_db.membru
    TO asistent@localhost
```

Prima instrucțiune acordă acces pentru citire la întregul tabel membru și configurează o parolă. Cea de-a doua instrucțiune adaugă privilegiul UPDATE, valabil însă numai pentru coloana data_expirare. Nu este necesar să specificați încă o dată parola, deoarece aceasta a fost precizată la prima instrucțiune.

Dacă doriți să acordați privilegii specifice coloanelor pentru mai mult de o coloană, specificați o listă de coloane separate prin virgulă. De exemplu, pentru a adăuga utilizatorului asistent privilegii UPDATE pentru câmpurile de adresă ale tabelului membru, folosiți instrucțiunea de mai jos. Noile privilegii vor fi adăugate la cele care există deja pentru utilizatorul respectiv:

```
GRANT UPDATE (strada,oras,stat,cod_postal) ON samp_db.membru
    TO asistent@localhost
```

Normal, nu doriți să acordați privilegii mai extinse decât cele cu adevărat necesare unui utilizator. Totuși, un motiv pentru a acorda privilegii relativ lejere asupra unei baze de date apare atunci când doriți ca utilizatorii dumneavoastră să poată crea tabele „ciornă" care să conțină rezultatele intermediare, dar nu doriți ca aceștia să procedeze astfel într-o bază de date care conține date pe care ei nu trebuie să le modifice. Puteți proceda astfel configurând o bază de date separată (o vom numi tmp) și acordând utilizatorilor toate privilegiile aferente acesteia. De exemplu, dacă doriți ca orice utilizator al gazdelor din domeniul mars.net să poată folosi baza de date tmp, puteți emite următoarea instrucțiune GRANT:

```
GRANT ALL ON tmp.* TO ""@%mars.net
```

După ce ați emis această instrucțiune, utilizatorii pot crea tabele în baza de date tmp și pot face referiri la acestea folosind nume de forma tmp.nume_tabel. (Caracterele "" din specificatorul de utilizator creează o intrare de utilizator anonim; orice utilizator este echivalent cu un nume de utilizator vid.)

Este necesar ca utilizatorul să aibă permisiunea de administrare a privilegiilor?

Puteți permite proprietarului unei baze de date să controleze accesul la baza de date, acordând proprietarului toate privilegiile aferente bazei de date și specificând în același timp și opțiunea WITH GRANT OPTION. De exemplu, dacă doriți ca utilizatorul alicia să se poată conecta de la orice gazdă din domeniul firma-mare.com și să administreze privilegiile aferente tuturor tabelelor din baza de date vanzari, puteți folosi următoarea instrucțiune GRANT:

```
GRANT ALL ON vanzari.*
    TO alicia@%.firma-mare.com IDENTIFIED BY "suc_de_mere"
    WITH GRANT OPTION
```

În consecință, clauza WITH GRANT OPTION vă permite să delegați unui alt utilizator drepturile de acordare a accesului. Rețineți că doi utilizatori cu privilegiul GRANT își pot acorda reciproc propriile lor privilegii. Dacă ați acordat unui utilizator numai privilegiul SELECT, dar un altul dispune de privilegiul GRANT, plus alte privilegii în afară de SELECT, cel de-al doilea utilizator îl poate „împuternici" pe primul.

Revocarea privilegiilor și eliminarea utilizatorilor

Pentru a elimina privilegiile unui utilizator, folosiți instrucțiunea REVOKE. Sintaxa acestei instrucțiuni este foarte similară aceleia a instrucțiunii GRANT, cu deosebirea că particula TO este înlocuită cu FROM și că nu mai există clauzele IDENTIFIED BY și WITH GRANT OPTION:

```
REVOKE privilegii (coloane) ON entitate FROM utilizator
```

Partea utilizator trebuie să coincidă cu partea utilizator a instrucțiunii GRANT originale pentru utilizatorul ale cărui privilegii doriți să le revocați. Partea privilegii nu trebuie să corespundă; puteți acorda privilegii cu o instrucțiune GRANT, după care puteți elimina numai o parte din ele cu instrucțiunea REVOKE.

Instrucțiunea REVOKE elimină numai privilegii, nu și utilizatori. O intrare aferentă unui utilizator rămâne în tabelul user chiar dacă revocați toate privilegiile utilizatorului. Aceasta înseamnă că utilizatorul are încă posibilitatea de a se conecta la server. Pentru a elimina în totalitate un utilizator, trebuie să ștergeți în mod explicit înregistrarea aferentă utilizatorului din tabelul user cu ajutorul unei instrucțiuni DELETE:

```
% mysql -u root mysql
mysql> DELETE FROM user
    -> WHERE User = "nume_utilizator" and Host ="nume_gazda";
mysql> FLUSH PRIVILEGES;
```

Instrucțiunea DELETE elimină intrarea aferentă utilizatorului, iar instrucțiunea FLUSH indică serverului să reîncarce tabelele de acordare. (Tabelele sunt reîncărcate automat atunci când folosiți GRANT sau REVOKE, nu și atunci când modificați direct tabelele de acordare.)

Secțiunea următoare descrie o situație care ilustrează un motiv care poate justifica ștergerea intrărilor din tabelul utilizator.

O enigmă cu privilegii (Partea I)

Iată un scenariu care apare frecvent în listele de corespondență MySQL: un nou administrator MySQL adaugă o intrare pentru un utilizator, dar folosește un nume de gazdă care este specificat folosind un model. De exemplu:

```
GRANT ALL ON samp_db.* TO fred@%.snake.net IDENTIFIED BY "cacao"
```

Intenția acestei instrucțiuni este de a permite utilizatorului fred să se conecteze de la orice gazdă situată în domeniul snake.net și să dispună de toate privilegiile aferente bazei de date samp_db. De fapt, fred se va putea conecta de la aceste gazde, *cu excepția însăși a gazdei serverului!* Când fred încearcă să se conecteze de la gazda serverului, încercarea eșuează cu un mesaj „accesul interzis", chiar dacă el specifică parola corectă.

Această situație se produce dacă tabelele dumneavoastră de acordare conțin intrările prestabilite instalate de scriptul de inițializare mysql_install_db. Motivul este acela că, atunci când serverul validează tentativa de conectare a lui fred, una dintre intrările de utilizator anonim capătă prioritate asupra intrării lui fred. Intrarea aferentă utilizatorului anonim impune ca utilizatorul să se conecteze fără parolă, deci se produce o neconcordanță între parole. Detalii suplimentare despre această pro-

blemă sunt furnizate în secțiunea „O enigmă cu privilegii (Partea a II-a)", din capitolul 12, „Securitate". Pentru moment, este suficient să amintim că o modalitate de a remedia problema constă în ștergerea intrărilor aferente utilizatorilor anonimi din tabelul user. Nu putem folosi REVOKE pentru această operație, deoarece instrucțiunea respectivă elimină numai privilegii. Pentru a elimina în totalitate aceste intrări, procedați astfel:

```
% mysql -u root mysql
mysql> DELETE FROM user WHERE User=" ";
mysql> FLUSH PRIVILEGES;
```

Acum, când fred va încerca să se conecteze de la gazda locală, va reuși.

Întreținerea fișierelor jurnal

Când serverul MySQL pornește, își examinează opțiunile în linie de comandă pentru a vedea dacă trebuie sau nu să efectueze consemnări în jurnal, respectiv deschide fișierele jurnal adecvate dacă trebuie să execute consemnări. Puteți indica serverului să genereze două categorii principale de jurnale:

- **Fișierul jurnal general.** Acesta raportează conexiunile clienților, interogările și alte diferite categorii de evenimente. Este util pentru urmărirea activității serverului, în speță pentru a determina persoana care se conectează, de unde se conectează și care au fost acțiunile întreprinse.
- **Fișierul de actualizare.** Acesta raportează interogări care modifică baza de date. Termenul „actualizare" se referă în acest context nu numai la instrucțiunile UPDATE, ci și la orice instrucțiune care modifică o bază de date. Din acest motiv, jurnalul conține o evidență a interogărilor precum DELETE, INSERT, REPLACE, CREATE TABLE, DROP TABLE, GRANT și REVOKE. Conținutul jurnalului de actualizare este scris ca instrucțiuni SQL, într-o formă care poate fi folosită drept date de intrare pentru mysql. Acest jurnal este util în conjuncție cu copiile de siguranță, dacă trebuie să restaurați tabelele în urma unei căderi. Puteți restaura o bază de date din fișierele care conțin copiile de siguranță, după care rulați din nou toate interogările care au modificat baza de date după executarea copiei de siguranță, folosind jurnalele de actualizare ca date de intrare pentru mysql. Astfel, puteți aduce tabelele la starea în care se aflau la momentul căderii.

Pentru a permite consemnarea în jurnal, folosiți opțiunea --log pentru a activa jurnalul general, respectiv opțiunea --log-update pentru a activa jurnalul de actualizare. Puteți specifica aceste opțiuni în linia de comandă pentru mysqld, safe_mysqld sau mysql.server, respectiv în grupul [mysqld] dintr-un fișier cu opțiuni. Când activați caracteristica de consemnare în jurnal, fișierele jurnal sunt scrise în mod prestabilit în catalogul de date al serverului.

Vă recomand să activați ambele tipuri de consemnare în jurnal atunci când începeți să utilizați MySQL pentru prima dată. După ce ați acumulat o oarecare experiență, doriți să utilizați numai jurnalul de actualizare, pentru a reduce necesitățile de spațiu pe disc.

După ce ați activat consemnarea în jurnal, trebuie să vă asigurați că discurile dumneavoastră nu se vor umple cu enorme cantități de informații consemnate în jurnal, mai ales dacă aveți un server solicitat, care prelucrează un mare număr de interogări. Puteți folosi rotația și expirarea fișierelor jurnal pentru a păstra ultimele câteva jurnale disponibile în calculator, dar simultan trebuie să preveniți creșterea necontrolată a dimensiunii fișierelor jurnal.

Rotația fișierelor jurnal se produce astfel: să presupunem că fișierul jurnal se numește jurnal. La prima rotație, jurnal primește numele jurnal.0, iar serverul începe să scrie un nou fișier jurnal. La a doua rotație, jurnal.0 primește numele jurnal.1, jurnal primește numele jurnal.0, iar serverul începe să scrie un alt fișier jurnal nou. În acest fel, fiecare fișier preia succesiv numele jurnal.0, jurnal.1 și așa mai departe. Când ajunge la un anumit punct al rotației, fișierul expiră.

Jurnalele de actualizare și instrucțiunile LOAD DATA

În prezent, când serverul execută o instrucțiune LOAD DATA, scrie în jurnalul de actualizare numai instrucțiunea în sine, nu și conținutul rândurilor care sunt încărcate. Aceasta înseamnă că o operație de recuperare care folosește jurnalele de actualizare va fi incompletă dacă fișierul de date rămâne inaccesibil. Pentru a asigura accesul la fișierul de date, nu-l ștergeți decât după ce a fost executată copia de siguranță a bazei de date.

Copii de siguranță ale sistemului

Jurnalele dumneavoastră de actualizare nu vor fi de nici un folos pentru recuperarea bazelor de date dacă o cădere a discului cauzează pierderea acestora. Nu uitați să executați sistematic copii de siguranță ale sistemului de fișiere. De asemenea, este o idee bună să scrieți jurnalele de actualizare pe un alt disc decât cel unde sunt stocate bazele dumneavoastră de date. Vezi Capitolul 10, „Catalogul de date MySQL", pentru instrucțiuni cu privire la modificarea amplasamentului fișierelor jurnal.

De exemplu, dacă rotiți jurnalele zilnic și doriți să păstrați activitatea dumneavoastră de o săptămână consemnată în jurnale, veți păstra jurnalele numerotate de la jurnal.0 la jurnal.6. La următoarea rotație, veți determina expirarea jurnalului jurnal.6, permițând suprascrierea conținutului fișierului jurnal.5 peste acesta, iar jurnal.5 va deveni noul jurnal.6. În acest mod, veți reține un număr de jurnale, dar veți împiedica și supraaglomerarea discului.

Frecvența rotației jurnalelor și numărul de jurnale vechi pe care le păstrați vor depinde de gradul de solicitare a serverului (serverele active generează mai multe informații în jurnale) și de cantitatea de spațiu pe disc pe care sunteți dispus să o alocați jurnalelor mai vechi. Când rotiți jurnalele generale, puteți indica serverului să închidă fișierul jurnal curent și să deschidă un fișier nou, folosind comanda mysqladmin flush-logs.

Un script care execută rotația jurnalelor generale poate arăta astfel (modificați-l astfel încât să conțină numele de bază al jurnalului dumneavoastră și locația catalogului de date, eventual și numărul jurnalelor vechi pe care doriți să le păstrați):

```
#! /bin/sh

base=numebaza
```

Continuare

Continuare

```
datadir=DATADIR

cd $datadir
mv $base.5 $base.6
mv $base.4 $base.5
mv $base.3 $base.4
mv $base.2 $base.3
mv $base.1 $base.2
mv $base.0 $base.1
mv $base $base.0
mysqladmin flush-logs
```

Cel mai indicat este să rulați acest script din contul `mysqladmin` pentru a vă asigura că fișierele jurnal se află în proprietatea acelui utilizator. Dacă vă păstrați parametrii de conexiune în fișierul cu opțiuni `.my.cnf`, nu trebuie să specificați nici un parametru în comanda `mysqladmin` din script. În caz contrar, poate doriți să configurați un utilizator cu privilegii limitate, care nu poate face nimic în afară de a emite comenzi de golire. Apoi, puteți insera în script parola acelui utilizator, cu un risc minim. Dacă doriți să procedați astfel, utilizatorul trebuie să aibă numai privilegiul `RELOAD`. De exemplu, pentru a denumi utilizatorul `flush` și pentru a-i atribui parola `flushpass`, folosiți următoarea instrucțiune `GRANT`:

```
GRANT RELOAD ON *.* TO flush@localhost IDENTIFIED BY "flushpass"
```

Când trebuie să efectuați o operație de golire într-un script, puteți proceda astfel:

```
mysqladmin -u flush -pflushpass flush-logs
```

Sub Linux, poate preferați să folosiți `logrotate` pentru a instala scriptul `mysql-log-rotate`, care este inclus în distribuția MySQL, în loc de a vă scrie propriul script. Examinați catalogul `support-files` al distribuției dumneavoastră MySQL, dacă scriptul `mysql-log-rotate` nu a fost instalat automat de un fișier RPM.

Rotația fișierelor jurnal este puțin diferită în cazul fișierelor de actualizare decât în cazul jurnalelor generale, datorită modalității în care serverul tratează numele fișierelor care conțin jurnalele de actualizare. Dacă indicați serverului să folosească un nume de fișier de tip jurnal de actualizare fără nici o extensie, precum `actualizare`, acesta va crea în mod automat nume de fișier pentru jurnalele de actualizare folosind secvența `actualizare.001`, `actualizare.002` și așa mai departe. Un nou fișier jurnal de actualizare este creat la pornirea serverului, respectiv la golirea jurnalelor. Dacă activați consemnarea în jurnale fără specificarea unui nume de fișier, serverul generează o secvență de fișiere jurnal de actualizare, folosind numele gazdei dumneavoastră ca nume de bază.

La expirarea unei secvențe de fișiere generate în acest mod, probabil că veți dori ca expirarea să se producă în funcție de vârstă (evaluată în raport cu momentul ultimei modificări), nu după nume. Motivul este acela că nu cunoașteți momentul când s-ar fi putut emite comenzile `flush-logs`, deci nu puteți conta pe un număr fix de jurnale de actualizare create în orice perioadă de timp dată. De exemplu, dacă realizați o copie de siguranță a tabelelor cu `mysqldump` și folosiți opțiunea `--flush-logs`, pentru fiecare copie de siguranță este creat un nou fișier din secvența numelor de fișiere de tip jurnal de actualizare.

Pentru jurnalele de actualizare ale căror nume de fișier secvențiale sunt generate automat de către server, un script de expirare bazat pe vârstă se poate prezenta astfel:

```
#! /bin/sh

find DATADIR -name "actualizare.[0-9]*" -type f -mtime +6 | xargs rm -f
mysqladmin flush-logs
```

Comanda `find` localizează și elimină fișierele jurnal de actualizare modificate ultima dată în urmă cu mai mult de o săptămână. Este important să folosiți argumentul `-name` pentru a verifica existența unei extensii numerice a numelui fișierului, pentru a evita eliminarea accidentală a unui tabel denumit `actualizare`.

Puteți cere serverului să folosească un nume de fișier fix pentru jurnalul de actualizare, dacă doriți. Acest fapt este util dacă doriți să rotiți jurnalele de actualizare în același mod ca și jurnalele generale. Pentru a folosi un nume fix al fișierului jurnal de actualizare, specificați un nume care include o extensie. De exemplu, puteți porni serverul cu opțiunea `--log-update=jurnal.actualizare` pentru a folosi numele `jurnal.actualizare`. Serverul va continua să închidă, respectiv să deschidă jurnalul de fiecare dată când este primită o comandă `flush-logs`, dar nu va genera un fișier nou la fiecare execuție a unei asemenea operații. În acest caz, scriptul de rotire a jurnalului folosit pentru jurnalele de actualizare va diferi de cel folosit pentru jurnalele generale numai în ceea ce privește numele de bază al fișierelor pe care le rotiți.

Probabil că veți dori să executați automat rotația și expirarea jurnalelor. Puteți proceda astfel folosind `cron`. Să presupunem că scripturile pentru rotația jurnalelor general și de actualizare se numesc `rotire-jurnale`, respectiv `rotire-jurnale-actualizare`, precum și că sunt instalate în catalogul `/usr/users/mysql/bin`. Deschideți sesiunea de lucru sub contul `mysqladmin` și editați fișierul `crontab` al utilizatorului `mysqladm` folosind această comandă:

```
% crontab -e
```

Această comandă vă va permite să editați o copie a fișierului dumneavoastră `crontab` curent (care poate fi vid, dacă nu ați mai efectuat în prealabil această operație). Adăugați la fișier linii care se prezintă astfel:

```
0 4 * * * /usr/users/mysqladm/bin/rotire-jurnale
0 4 * * * /usr/users/mysqladm/bin/rotire-jurnale-actualizare
```

Această intrare indică utilitarului `cron` să ruleze scriptul la ora 4 în fiecare dimineață. Puteți modifica ora de rulare după dorință; pentru instrucțiuni, examinați pagina de manual referitoare la comanda `crontab`.

Realizarea copiilor de siguranță și a copiilor obișnuite ale bazelor de date

Este important să realizați copii de siguranță ale bazelor dumneavoastră de date, în caz că acestea se pierd sau sunt deteriorate. Dacă se produce o cădere a sistemului, doriți să vă puteți restaura tabelele la starea în care se găseau în momentul căderii sistemului, cu o pierdere de date minimă. De asemenea, este posibil ca un utilizator care emite o instrucțiune `DROP DATABASE` sau `DROP TABLE` nechibzuită să apară la ușa dumneavoastră, rugându-vă să-i recuperați datele. Uneori, administratorul MySQL este cel care cauzează paguba. Este cunoscut faptul că administratorii au deteriorat fișiere tabel încercând să le editeze în mod direct, folosind un editor precum `vi` sau `emacs`. Nu se procedează așa cu un tabel!

Cele două metode principale pentru executarea copiilor de siguranță ale bazelor de date constau în utilizarea programului `mysqldump` sau în copierea directă a fișierelor de baze de date (de exemplu folosind `cp`, `tar` sau `cpio`). Fiecare metodă își are propriile avantaje și dezavantaje:

- `mysqldump` operează în cooperare cu serverul MySQL. Metodele de copiere directă sunt externe față de server și trebuie să luați măsuri pentru a vă asigura că nici un client nu modifică tabelele în timp ce dumneavoastră le copiați. Este aceeași problemă care apare dacă încercați să folosiți copii de siguranță ale sistemului de fișiere pentru executarea copiilor de siguranță ale bazelor de date: dacă un tabel din baza de date este actualizat în timpul execuției copiilor de siguranță a sistemului de fișiere, fișierele tabel care intră în copia de siguranță se află într-o stare inconsecventă și sunt inutile pentru restaurarea ulterioară a tabelului. Diferența între copiile de siguranță ale sistemelor de fișiere și fișierele obținute prin copiere directă este aceea că, în ultimul caz, aveți controlul asupra planificării orare a execuției copiei de siguranță, deci puteți lua măsuri pentru a vă asigura că serverul lasă tabelul în pace.
- `mysqldump` este mai lent decât tehnicile de copiere directă.
- `mysqldump` generează fișiere text care sunt portabile pe alte calculatoare, chiar și pe acelea cu o arhitectură hardware diferită. Fișierele obținute prin copiere directă nu sunt portabile pe alte mașini decât dacă tabelele pe care le copiați folosesc formatul de stocare MyISAM. Tabelele ISAM pot fi copiate numai între calculatoare cu arhitecturi hardware asemănătoare. De exemplu, este permisă copierea fișierelor dintr-un sistem Solaris instalat pe SPARC într-un sistem Solaris instalat pe SPARC, dar copierea fișierelor dintr-un sistem Solaris instalat pe SPARC pe un sistem Solaris instalat pe Intel sau invers nu va reuși. Formatul de stocare a tabelelor MyISAM introdus în MySQL 3.23 rezolvă această problemă, deoarece formatul este independent de mașină. Ca atare, fișierele copiate direct pot fi mutate pe o mașină cu o altă arhitectură hardware, dacă sunt respectate două condiții: mașina pe care se copiază trebuie să ruleze MySQL 3.23 sau o versiune ulterioară, iar fișierele trebuie să reprezinte tabele MyISAM, nu ISAM.

Indiferent de metoda de realizare a copiilor de siguranță pe care o alegeți, există anumite principii pe care trebuie să le respectați pentru a obține cele mai bune rezultate, în caz că trebuie vreodată să restaurați conținutul bazei de date:

- Executați sistematic copii de siguranță. Stabiliți un orar de execuție a copiilor de siguranță și respectați-l.
- Indicați serverului să consemneze actualizările în jurnal (vezi secțiunea „Întreținerea fișierelor jurnal"). Jurnalele de actualizare sunt utile atunci când trebuie să restaurați o bază de date după o cădere. După ce vă folosiți fișierele copie de siguranță pentru a restaura baza de date la starea în care se afla la momentul execuției copiei de siguranță, puteți aplica din nou schimbările efectuate după executarea copiei de siguranță, prin rularea interogărilor din jurnalele de actualizare. Astfel, tabelele din baza de date sunt readuse la starea în care se aflau la momentul căderii.

  În limbajul copiilor de siguranță ale sistemelor de fișiere, fișierele copie de siguranță ale bazelor de date reprezintă copii complete, iar jurnalele de actualizare reprezintă copii incrementale.

- Folosiți o schemă de denumire a fișierelor consecventă și inteligibilă. Nume precum `copie1`, `copie2` etc. nu sunt deosebit de semnificative. Când vine momentul să efectuați restaurarea, veți irosi timp încercând să aflați care este conținutul fișierelor. Este util să construiți numele fișierelor care conțin copiile de siguranță folosind numele bazelor de date și datele calendaristice. De exemplu:

```
% mysqldump samp_db > /usr/archives/mysql/samp_db.1999-10-02
% mysqldump menagerie > /usr/archives/mysql/menagerie.1999-10-02
```

  Puteți comprima fișierele copie de siguranță după ce le generați. Copiilor de siguranță le place să fie mari, la urma urmelor! De asemenea, va trebui să provocați expirarea fișierelor dumneavoastră copie de siguranță, pentru a le împiedica să vă umple discul, așa cum procedați la expirarea fișierelor jurnal. Pentru informații cu privire la expirarea fișierelor jurnal, vezi secțiunea „Întreținerea fișierelor jurnal". Puteți folosi aceleași tehnici pentru expirarea fișierelor copie de siguranță.
- Creați copii de siguranță ale copiilor de siguranță, folosind copii de siguranță ale sistemelor de fișiere. Dacă se produce o cădere completă, care „rade" complet nu numai catalogul de date, dar și unitatea de disc care conține copiile de siguranță ale bazelor de date, atunci ați dat de belea. Executați și copii de siguranță ale jurnalelor de actualizare.
- Amplasați fișierele copie de siguranță într-un sistem de fișiere diferit de acela pe care îl folosiți pentru bazele dumneavoastră de date. Astfel, se reduce probabilitatea de umplere a sistemului de fișiere care conține catalogul de date, ca rezultat al generării copiilor de siguranță.

Tehnicile care se folosesc pentru crearea copiilor de siguranță sunt de asemenea utile pentru copierea bazelor de date într-un alt server. Cel mai frecvent, o bază de date este transferată pe un server care rulează pe o altă gazdă, dar puteți transfera date pe un alt server, care rulează pe aceeași gazdă. Puteți proceda astfel dacă rulați un server pentru o versiune mai recentă a sistemului MySQL și doriți să-l testați cu unele date reale provenite din serverul dumneavoastră de producție. Sau v-ați procurat un calculator nou și doriți să vă mutați toate bazele de date în acesta.

Utilizarea programului `mysqldump` pentru executarea copiilor normale și de siguranță ale bazelor de date

Când folosiți programul `mysqldump` pentru a genera fișiere copie de siguranță ale bazelor de date, conținutul fișierului constă, în mod prestabilit, din instrucțiunile `CREATE TABLE` care au creat tabelele afișate, respectiv instrucțiunile `INSERT` care conțin datele inserate în rândurile tabelelor. Cu alte cuvinte, `mysqldump` creează date de ieșire care se pot folosi ulterior ca date de intrare ale programului `mysql` pentru re-crearea bazei de date.

Puteți transfera o întreagă bază de date într-un singur fișier text, astfel:

```
% mysqldump samp_db  /usr/archives/mysql/samp_db.1999-10-02
```

Începutul fișierului de ieșire se prezintă astfel:

```
# MySQL dump 6.0
#
# Host: localhost          Database: samp_db
```

Continuare

Continuare

```
# ----------------------------------------------------
# Server version                3.23.2-alpha-log
#
# Table structure for table 'absente'
#
CREATE TABLE absente (
  elev_id int(10) unsigned DEFAULT '0' NOT NULL,
  data date DEFAULT '0000-00-00' NOT NULL,
  PRIMARY KEY (elev_id,data)
);
#
# Dumping data for table 'absente'
#
INSERT INTO absente VALUES (3,'1999-09-03');
INSERT INTO absente VALUES (5,'1999-09-03');
INSERT INTO absente VALUES (10,'1999-09-06');
...
```

Restul fișierului este alcătuit din alte instrucțiuni `INSERT` și `CREATE TABLE`.

Dacă doriți să comprimați copia de siguranță la generarea acesteia, folosiți în schimb o comandă ca aceasta:

```
% mysqldump samp_db | gzip > /usr/archives/mysql/samp_db.1999-10-02.gz
```

Dacă aveți o bază de date de foarte mari dimensiuni, fișierul de ieșire va fi de asemenea foarte mare și poate fi dificil de gestionat. Dacă preferați, puteți transfera conținutul tabelelor individuale denumindu-le în funcție de numele bazei de date în linia de comandă `mysqldump`. Astfel, fișierul final este partiționat în fișiere mai mici și mai ușor de gestionat. Exemplul următor ne prezintă modul de transferare a unora dintre tabelele din baza de date `samp_db` în fișiere separate:

```
% mysqldump samp_db elev puncte eveniment absente > catalog.sql
% mysqldump samp_db membru presedinte > liga-istorica.sql
```

Dacă generați fișiere copii de siguranță care sunt destinate a fi utilizate pentru reîmprospătarea periodică a conținutului unei alte baze de date, puteți să folosiți opțiunea `--add-drop-table`. Această opțiune indică programului `mysqldump` să scrie în fișierul copie de siguranță instrucțiuni `DROP TABLE IF EXISTS`. Apoi, când luați fișierul copie de siguranță și îl încărcați în a doua bază de date, nu veți obține un mesaj de eroare dacă tabelele există deja. Dacă rulați un al doilea server de test, puteți folosi această tehnică pentru a-l reîncărca periodic cu o copie a datelor situate în bazele de date din serverul dumneavoastră de producție.

Dacă transferați baza de date într-un fișier pentru a o putea deplasa într-un alt server, nici măcar nu trebuie să creați fișiere copie de siguranță. Verificați dacă baza de date există la cealaltă gazdă, apoi transferați baza de date folosind un canal, astfel încât `mysql` să citească direct datele de ieșire ale programului `mysqldump`. De exemplu, dacă doriți să copiați baza de date `samp_db` din sistemul `pit-viper.snake.net` în calculatorul `boa.snake.net`, acest lucru se poate face cu ușurință astfel:


```
% mysqladmin -h boa.snake.net create samp_db
% mysqldump samp_db | mysql -h boa.snake.net samp_db
```

Mai târziu, dacă doriți să reîmprospătați baza de date respectivă situată pe `boa.snake.net`, ignorați comanda `mysqladmin`, dar adăugați opțiunea `--add-drop-table` la `mysqldump` pentru a evita obținerea unor mesaje de eroare referitoare la tabele care există deja:

```
% mysqldump --add-drop-table samp_db | mysql -h boa.snake.net samp_db
```

Alte opțiuni utile ale comenzii `mysqldump` includ următoarele:

- Combinația dintre `--flush-logs` și `--lock-tables` este utilă pentru stabilirea de puncte de reper în baza dumneavoastră de date. Opțiunea `--lock-tables` blochează toate tabelele pe care le transferați, iar `--flush-logs` închide și redeschide fișierul jurnal de actualizare. Dacă generați jurnale de actualizare în secvență, noul jurnal de actualizare va conține numai interogări care modifică bazele de date începând din momentul execuției copiei de siguranță. Astfel, jurnalul dumneavoastră de actualizare este un punct de reper la momentul execuției copiei de siguranță. (Totuși, blocarea tuturor tabelelor nu este o idee atât de bună pentru accesul clienților în timpul execuției copiei de siguranță, dacă aveți clienți care trebuie să execute actualizări.)

  Dacă folosiți `--flush-logs` pentru a crea în jurnalele de actualizare un punct de reper pentru momentul execuției copiei de siguranță, probabil că este cel mai indicat să transferați întreaga bază de date. Dacă transferați fișiere individuale, este mai dificil să sincronizați punctele de reper ale jurnalelor de actualizare cu fișierele copie de siguranță. În timpul operațiilor de restaurare, de regulă extrageți conținutul jurnalelor de actualizare pentru fiecare bază de date în parte. Nu există nici o opțiune de extragere a actualizărilor pentru tabele individuale, deci va trebui să le extrageți personal.

- În mod prestabilit, `mysqldump` citește întregul conținut al unui tabel în memorie înainte de a-l scrie. Acest fapt nu este absolut necesar și este o cale aproape sigură spre eșec, dacă aveți tabele de dimensiuni foarte mari. Puteți folosi opțiunea `--quick` pentru a indica programului `mysqldump` să scrie fiecare rând pe măsură ce este regăsit. Pentru a optimiza în continuare procesul de transfer, folosiți `--opt` în loc de `--quick`. Opțiunea `--opt` activează alte opțiuni care accelerează transferul datelor și recitirea lor.

  Executarea copiilor de siguranță folosind opțiunea `--opt` este, probabil, metoda cea mai comună, datorită avantajelor determinate de viteza de executare a copiilor. Rețineți, totuși, că opțiunea `--opt` își are prețul său; această opțiune optimizează numai procedura de creare a copiilor de siguranță, nu accesul altor clienți la baza de date. Opțiunea `--opt` împiedică pe orice utilizator să actualizeze vreunul din tabelele pe care le transferați, prin blocarea simultană a tuturor tabelelor. Puteți constata cu ușurință efectul acestei opțiuni asupra accesului general la baza de date. Încercați numai să executați o copie de siguranță în momentul din zi când baza dumneavoastră de date este cel mai puternic solicitată. Nu va dura prea mult și telefonul dumneavoastră va începe să fie bombardat de persoane care vor dori să știe ce s-a întâmplat. (V-aș fi recunoscător dacă v-ați abține să mă întrebați cum se întâmplă că știu toate astea...)

- O opțiune cu un efect oarecum opus lui `--opt` este `--delayed`. Această opțiune determină programul `mysqldump` să scrie instrucțiuni `INSERT DELAYED` și nu instrucțiuni `INSERT`. Dacă încărcați un fișier de date într-o altă bază de date și doriți să reduceți la

minimum impactul acestei operații asupra altor interogări care pot avea loc în baza de date respectivă, opțiunea `--delayed` este categoric utilă.

- Opțiunea `--compress` este utilă atunci când copiați o bază de date într-un alt calculator, deoarece reduce numărul de octeți care parcurg rețeaua. Iată un exemplu; observați că opțiunea `--compress` este dată pentru programul care comunică cu serverul situat pe gazda aflată la distanță, nu pentru cel care comunică cu gazda locală:

```
% mysqldump --opt samp_db | mysql --compress -h boa.snake.net samp_db
```

Programul `mysqldump` are numeroase alte opțiuni. Pentru mai multe informații, consultați Anexa E.

Utilizarea metodelor de creare prin copiere directă a copiilor obișnuite și de siguranță a bazelor de date

O altă modalitate de a crea o copie de siguranță a unei baze de date sau a unor tabele, care nu implică utilizarea programului `mysqldump`, este copierea directă a fișierelor tabel. În mod caracteristic, această operație se execută folosind utilitare precum `cp`, `tar` sau `cpio`. Exemplele din această secțiune vor folosi programul `cp`.

Când folosiți o metodă de realizare a copiilor de siguranță prin copiere directă, trebuie să vă asigurați că tabelele nu sunt folosite. Dacă serverul modifică un tabel în timp ce dumneavoastră îl copiați, copiile nu vor avea nici o valoare.

Cea mai bună modalitate de a asigura integritatea copiilor dumneavoastră este de a opri serverul, a copia fișierele și apoi de a reporni serverul. Dacă nu doriți să opriți serverul, trebuie să consultați capitolul 13. Examinați tehnicile descrise acolo pentru blocarea serverului în timpul verificării tabelelor. Dacă serverul rulează, aceleași restricții se aplică asupra copierii fișierelor, iar dumneavoastră trebuie să folosiți același protocol de blocare pentru a reduce serverul la tăcere.

Presupunând fie că serverul este oprit, fie că ați blocat tabelele pe care doriți să le copiați, exemplul următor prezintă modul de execuție a copiei de siguranță a întregii baze de date `samp_db` într-un catalog pentru copii de siguranță (`DATADIR` reprezintă catalogul de date al serverului):

```
% cd DATADIR
% cp -r samp_db /usr/archive/mysql
```

Copiile de siguranță ale tabelelor individuale se pot executa astfel:

```
% cd DATADIR/samp_db
% cp membru.* /usr/archive/mysql/samp_db
% cp puncte.* /usr/archive/mysql/samp_db
...
```

Când ați terminat execuția copiilor de siguranță, puteți reporni serverul (dacă l-ați oprit), respectiv puteți elimina blocările aplicate asupra tabelelor (dacă ați lăsat serverul în funcțiune).

În vederea utilizării fișierelor obținute prin copiere directă pentru copierea unei baze de date de la un calculator la altul, copiați fișierele în catalogul de baze de date adecvat din cealaltă gazdă de server. Verificați dacă fișierele sunt destinate tabelelor MyISAM sau dacă ambele calculatoare au aceeași arhitectură hardware. În caz contrar, tabelele dumneavoastră vor avea un conținut foarte ciudat, odată copiate pe cea de-a doua gazdă. De asemenea, trebuie să vă asigurați că serverul de la a doua gazdă nu încearcă să obțină acces la tabele în timp ce dumneavoastră le instalați.

Replicarea bazelor de date

Termenul de „replicare" se poate reduce la copierea unei baze de date pe un alt server sau poate implica actualizarea directă a unei baze de date secundare simultan cu efectuarea de modificări asupra conținutului bazei de date master. Dacă nu doriți decât să copiați o bază de date pe un alt server, puteți folosi pentru aceasta comenzile care au fost deja prezentate anterior. Suportul pentru replicarea bazată pe actualizarea „în direct" a început să apară din versiunea 3.23 a sistemului MySQL. Această caracteristică este încă în faza de început, deci nu voi spune mai mult decât că trebuie să urmăriți versiunile curente, pe măsură ce acestea sunt lansate, pentru a vedea evoluțiile care au loc, dacă subiectul vă interesează.

Utilizarea copiilor de siguranță pentru refacerea datelor

Deteriorarea unei baze de date se produce dintr-un număr de motive și variază ca amploare. Dacă aveți noroc, s-ar putea să vă alegeți cu pagube minore la un tabel-două (de exemplu, dacă sistemul dumneavoastră își întrerupe pentru scurt timp activitatea datorită unei pene de tensiune). Dacă nu sunteți atât de norocos, s-ar putea să fiți obligat să înlocuiți întregul catalog de date (de exemplu dacă un disc a suferit o cădere și a „luat cu el" catalogul dumneavoastră de date). Refacerea este de asemenea necesară și în alte situații, cum ar fi ștergerea accidentală a bazelor de date sau a tabelelor, respectiv ștergerea conținutului unui tabel de către utilizatori. Indiferent de motivul acestor evenimente nefericite, va trebui să executați unele operații de refacere.

Dacă tabelele au suferit deteriorări, dar nu s-au pierdut, încercați să le reparați folosind `myisamchk` sau `isamchk`. S-ar putea să nu fiți obligat să folosiți deloc copii de siguranță, dacă paguba poate fi remediată de utilitarele de reparare. Procedura pentru repararea tabelelor este descrisă în Capitolul 13. Dacă tabelele sunt pierdute sau nu se pot repara, va trebui să le restaurați.

Procedurile de refacere implică două surse de informații: fișierele copie de siguranță și jurnalele de actualizare. Fișierele copie de siguranță restaurează tabelele în starea în care se găseau în momentul execuției copiei de siguranță. Totuși, de obicei tabelele mai suferă modificări între momentul execuției copiei și momentul apariției problemelor. Jurnalele de actualizare conțin interogările folosite pentru a opera aceste modificări. Puteți repeta interogările folosind jurnalele de actualizare ca date de intrare pentru `mysql`. (De aceea trebuie să activați caracteristica de consemnare în jurnalul de actualizare. Dacă încă nu ați procedat astfel, trebuie să o faceți chiar acum și să generați o nouă copie de siguranță înainte de a continua lectura.)

Procedurile de refacere variază în funcție de cantitatea de informații pe care trebuie să o restaurați. De fapt, restaurarea unei întregi baze de date poate fi mai simplă decât restaurarea unui singur tabel, deoarece aplicarea jurnalelor de actualizare este mai simplă pentru o bază de date decât pentru un tabel.

Refacerea unei întregi baze de date

Mai întâi, dacă baza de date pe care doriți să o refaceți este baza de date `mysql` care conține tabelele de acordare, va trebui să rulați serverul folosind opțiunea `--skip-grant-tables`. În caz contrar, serverul va reclama că nu poate găsi tabelele de acordare. După ce ați restaurat tabelele, executați comanda `mysqladmin flush-privileges` pentru a indica serverului să încarce tabelele de acordare și să înceapă să le folosească.

- Copiați undeva în altă parte conținutul catalogului bazei de date. Veți avea nevoie de acesta mai târziu – de exemplu, pentru a executa examinări post-mortem ale „cadavrelor" tabelelor distruse.
- Încărcați din nou baza de date, folosind cele mai recente copii de siguranță. Dacă folosiți fișiere generate de `mysqldump`, folosiți-le ca date de intrare pentru `mysql`. Dacă folosiți fișiere care au fost copiate direct din baza de date (de exemplu folosind programele `tar` sau `cp`), copiați-le direct în catalogul de baze de date. În acest caz, totuși, trebuie să opriți serverul înainte de a copia fișierele; veți porni serverul după aceea.
- Folosiți jurnalele de actualizare pentru a repeta interogările care au modificat tabelele din baza de date ulterior momentului execuției copiei de siguranță. Folosiți orice jurnal de actualizare aplicabil ca date de intrare pentru `mysql`. Specificați opțiunea `--one-database` astfel încât `mysql` să execute interogările numai pentru baza de date care sunteți interesat să o refaceți. Dacă știți că trebuie să aplicați toate fișierele jurnalelor de actualizare, puteți folosi această comandă în catalogul care conține jurnalele:

```
% ls -t -r -1 update.[0-9]* | xargs cat | mysql --one-database
    nume_baza_de_date
```

Comanda `ls` produce o listă (pe o singură coloană) a fișierelor jurnalelor de actualizare, sortate conform ordinii în care au fost generate de server. (Rețineți că, dacă modificați oricare dintre fișiere, veți modifica ordinea de sortare, ceea ce va avea ca efect aplicarea jurnalelor de actualizare într-o ordine greșită.)

Probabil că nu veți avea de aplicat decât câteva jurnale de actualizare. De exemplu, dacă jurnalele generate de la momentul execuției copiei de siguranță se numesc `update.392`, `update.393` și așa mai departe, puteți rula din nou comenzile pe care le conțin, după cum urmează:

```
% mysql --one-database nume_baza_de_date < update.392
% mysql --one-database nume_baza_de_date < update.393
...
```

Dacă executați refaceri și folosiți jurnale de actualizare pentru restaurarea informațiilor pierdute datorită unei instrucțiuni `DROP DATABASE`, `DROP TABLE` sau `DELETE` necugetate, nu uitați să eliminați instrucțiunea respectivă din jurnalul de actualizare în care apare instrucțiunea, înainte de a aplica jurnalul!

Refacerea tabelelor individuale

Refacerea tabelelor individuale este mai dificilă. Dacă aveți o copie de siguranță generată de `mysqldump` și acesteia îi lipsesc exact datele pentru tabelul care vă interesează, va trebui să extrageți liniile relevante și să le folosiți ca date de intrare pentru `mysql`. Aceasta este partea ușoară. Partea dificilă constă în extragerea acelor componente ale jurnalului de actualizare care se aplică chiar acelui tabel. Pentru această operație se poate folosi utilitarul `mysql_find_rows`, care poate extrage interogări pe mai multe linii din jurnalele de actualizare.

O altă posibilitate constă în restaurarea întregii baze de date folosind un alt server, apoi copiați fișierele pentru tabelul dorit în baza de date originală. Această operație poate fi chiar mai ușoară! Verificați ca serverul pentru baza de date originală să fie oprit atunci când copiați fișierele înapoi în catalogul bazei de date.

Ajustarea serverului

Serverul MySQL are numeroși parametri (variabile) care controlează modul său de operare. Dacă valorile prestabilite ale parametrilor nu sunt corespunzătoare, le puteți transforma în valori mai adecvate pentru mediul în care rulează serverul dumneavoastră. De exemplu, dacă dispuneți de o mare cantitate de memorie, puteți cere serverului să folosească buffere mai mari pentru operațiile cu discul și operațiile de indexare. Astfel, se vor stoca mai multe informații în memorie și se va reduce numărul necesar de operații de acces la disc. Dacă sistemul dumneavoastră este mai modest, puteți cere serverului să folosească buffere mai mici, pentru a preveni aglomerarea resurselor sistemului în detrimentul altor procese.

Valorile curente ale variabilelor serverului pot fi examinate prin executarea comenzii `mysqladmin variables`. Variabilele pot fi configurate cu ajutorul opțiunii `--set-variable nume_variabila=valoare` din linia de comandă (opțiune echivalentă cu `-O nume_variabila=valoare`). Dacă doriți să configurați numeroase variabile, folosiți mai multe opțiuni `--set-variable`. De asemenea, puteți configura variabilele din grupul `[mysqld]` al unui fișier cu opțiuni, folosind următoarea sintaxă:

```
set-variable=nume_variabila=valoare
```

O listă completă a variabilelor serverului este dată în Anexa E, în cadrul intrării aferente programului `mysqld`. Variabilele mai utile pentru ajustarea performanțelor sunt prezentate în lista următoare. Detalii suplimentare despre acest subiect pot fi găsite în capitolul „Utilizarea sistemului MySQL la maximum de performanță" din manualul de referință MySQL.

- `back_log`

  Numărul de cereri de conexiune recepționate din partea clienților care pot fi așezate la coadă în timpul prelucrării cererilor provenite de la clienții curenți. Dacă aveți un sit foarte solicitat, este necesar să măriți valoarea acestei variabile.
- `delayed_queue_size`

  Această variabilă controlează numărul de rânduri al instrucțiunilor `INSERT DELAYED` care pot fi așezate la coadă. În cazul în care coada este plină, instrucțiunile `INSERT DELAYED` se blochează până când în coadă se creează un spațiu, ceea ce îi împiedică pe clienții care emit aceste instrucțiuni să continue. Dacă aveți mai mulți clienți care execută acest gen de instrucțiuni `INSERT` și descoperiți că sunt blocați, creșterea valorii acestei variabile va permite mai multor clienți să continue după o perioadă mai scurtă de timp. (Instrucțiunea `INSERT DELAYED` este discutată în secțiunea „Probleme de planificare și blocare" din capitolul 4, „Optimizarea interogărilor")

- `flush_time`

  Dacă sistemul dumneavoastră are probleme şi tinde să se blocheze sau să reîncarce frecvent sistemul de operare, configurarea acestei variabile la o valoare diferită de zero determină serverul să golească zone cache aferentă tabelelor la fiecare `flush_time` secunde. Scrierea modificărilor din tabele în acest mod degradează performanţele, dar poate reduce posibilitatea de deteriorare a tabelelor sau de pierdere a datelor.

  Sub Windows, puteţi porni serverul cu opţiunea `--flush` în linia de comandă, pentru a forţa eliminarea modificărilor din tabele după fiecare actualizare.

- `key_buffer_size`

  Dimensiunea bufferului folosit pentru stocarea blocurilor de index. Crearea şi modificarea indexurilor se desfăşoară mai rapid dacă măriţi valoarea acestei variabile. Valorile mai mari ale variabilei măresc probabilitatea ca MySQL să găsească valorile cheie în memorie, ceea ce reduce numărul de operaţii cu discul necesare pentru prelucrarea indexului.

  Această variabilă se numeşte `key_buffer` în versiunile MySQL anterioare versiunii 3.23. De la versiunea 3.23, MySQL recunoaşte ambele nume.

- `max_allowed_packet`

  Dimensiunea maximă la care poate ajunge bufferul folosit pentru comunicaţiile cu clienţii. Dacă aveţi clienţi care trimit valori `BLOB` sau `TEXT` de mari dimensiuni, este potenţial necesară creşterea valorii acestei variabile.

  În prezent, clienţii folosesc o dimensiune prestabilită a bufferului de 24MB. Dacă aveţi clienţi mai vechi, care folosesc un buffer mai mic, poate că este nevoie să măriţi dimensiunea bufferului client. De exemplu, pentru a se specifica o limită de 24MB pentru pachet se poate proceda astfel:

  ```
  mysql --set-variable max_allowed_packet=24M
  ```

- `max_connections`

  Numărul maxim de conexiuni simultane cu clienţii, permis de server. Dacă serverul dumneavoastră este solicitat, puteţi fi obligat să măriţi această valoare. De exemplu, dacă serverul dumneavoastră MySQL este folosit de către serverul Web pentru a prelucra interogări generate de scripturi DBI sau PHP, iar traficul de Web al serverului are valori foarte ridicate, s-ar putea ca vizitatorii sitului dumneavoastră să-şi vadă cererile refuzate dacă această variabilă are o valoare prea redusă.

- `table_cache`

  Dimensiunea zonei cache pentru tabele. Creşterea acestei valori permite programului `mysqld` să păstreze mai multe tabele deschise simultan şi reduce numărul de operaţii necesare de deschidere, respectiv închidere a fişierelor.

Dacă măriţi valorile variabilelor `max_connection` sau `table_cache`, serverul va impune un număr mai mare de descriptori de fişier, ceea ce poate cauza probleme cu limitele sistemului de operare legate de numărul de descriptori de fişier per proces, caz în care va trebui să măriţi limita sau să găsiţi o altă soluţie. Există diverse proceduri pentru creşterea limitei numărului de descriptori de fişier. Puteţi executa această operaţie la rulare, folosind comanda `ulimit` în scriptul pe care îl utilizaţi pentru a porni serverul, sau puteţi fi obligat să vă reconfiguraţi sistemul. Unele sisteme pot fi configurate pur şi simplu prin editarea unui fişier de descriere a sistemului, urmată de reîncărcarea sistemului de operare. În alte situaţii, trebuie să editaţi un fişier de deschidere a nucleului şi să reconstruiţi nucleul. Pentru a vedea cum puteţi proceda, consultaţi documentaţia aferentă sistemului dumneavoastră.

O modalitate de a ignora limitele legate de descriptorii de fişier per proces este divizarea catalogului dumneavoastră de date în mai multe cataloage de date şi rularea mai multor servere. Astfel, numărul descriptorilor de fişiere disponibili se înmulţeşte efectiv cu numărul de servere care rulează. Pe de altă parte, alte complicaţii vă pot cauza probleme. Pentru a preciza numai două dintre acestea, nu puteţi obţine de la un singur server accesul la bazele de date din cataloage de date diferite şi puteţi fi obligat să replicaţi privilegiile din tabelele de acordare pe mai multe servere, pentru utilizatorii care au nevoie de acces la mai mult de un singur server.

Uneori, în speranţa îmbunătăţirii performanţelor, administratorii măresc valorile a două variabile: `record_buffer` şi `sort_buffer`. Aceste buffere sunt folosite în timpul operaţiilor de unire şi de sortare, dar valorile sunt specificate pentru fiecare conexiune în parte. Cu alte cuvinte, fiecare client beneficiază de propriile sale buffere. Dacă valorile acestor variabile devin foarte mari, performanţele pot avea de suferit, datorită consumului exorbitant de resurse ale sistemului. Dacă doriţi să modificaţi aceste variabile, executaţi comanda `mysqladmin variables` pentru a vedea care sunt valorile lor curente, iar apoi ajustaţi valorile incremental. Acest procedeu vă va permite să evaluaţi efectul modificării, iar posibilitatea unei degradări severe a performanţelor sistemului va fi mai redusă.

Rularea mai multor servere

Majoritatea dintre noi rulăm un singur server MySQL pe un calculator dat, dar există situaţii în care rularea mai multor servere se poate dovedi utilă:

- Doriţi să testaţi o nouă versiune a serverului, în timp ce serverul dumneavoastră de producţie funcţionează. În acest caz, veţi rula versiuni de server diferite.
- În mod caracteristic, sistemele de operare impun limitări per proces legate de numărul de descriptori ai fişierelor deschise. Dacă sistemul dumneavoastră nu facilitează creşterea acestei limite, rularea de servere multiple este o soluţie de a contracara acest dezavantaj. (De exemplu, creşterea limitei poate necesita recompilarea nucleului, iar dumneavoastră s-ar putea să nu puteţi executa această operaţie dacă nu sunteţi însărcinat cu administrarea sistemului.) În acest caz, puteţi rula mai multe instanţe ale aceleiaşi versiuni de server.
- Furnizorii de servicii Internet furnizează deseori clienţilor propria instalare a sistemului MySQL, ceea ce impune utilizarea de servere separate. În acest caz, puteţi rula mai multe instanţe ale aceleiaşi versiuni, respectiv versiuni diferite, dacă diferiţi clienţi doresc versiuni diferite ale sistemului MySQL.

Natural, rularea mai multor servere este mai complicată decât rularea unuia singur. Dacă instalaţi mai multe versiuni, nu le puteţi pur şi simplu instala pe toate în acelaşi loc. Când serverele rulează, anumiţi parametri trebuie să fie sau au tendinţa de a fi unici pentru fiecare server. Unii dintre aceştia includ locul de instalare a serverului, calea de acces spre

catalogul său de date, numele căii de acces spre portul TCP/IP, respectiv spre soclul de domeniu UNIX, precum și contul UNIX folosit pentru rularea serverului (dacă nu rulați toate serverele sub același cont). Dacă decideți să rulați mai multe servere, nu uitați să rețineți bine parametrii pe care îi folosiți, pentru a nu pierde controlul asupra situației.

Configurarea și actualizarea serverelor multiple

Dacă intenționați să rulați diferite versiuni ale serverului, în loc de mai multe instanțe ale aceluiași server, trebuie să le instalați în locații diferite. Dacă instalați distribuții binare (nu RPM), acestea vor fi instalate sub un nume de catalog care include numărul de versiune. Dacă instalați dintr-o distribuție sursă, cea mai simplă modalitate de a păstra separate diferite distribuții este de a folosi opțiunea `--with-prefix` atunci când rulați `configure` în timpul procesului de instalare MySQL pentru fiecare versiune. Astfel, totul va fi instalat într-un singur catalog și puteți stabili o legătură între acel catalog și numărul versiunii de distribuție. De exemplu, puteți configura o distribuție MySQL după cum urmează, unde *versiune* este numărul versiunii MySQL:

```
% ./configure --with-prefix=/usr/local/mysql-versiune
```

Opțiunea `--with-prefix` va determina, de asemenea, un catalog de date unic pentru server. Puteți dori să adăugați opțiuni pentru configurarea altor valori specifice serverului, cum ar fi numărul de port TCP/IP și calea de acces a soclului (`--with-tcp-port` și `--with-unix-socket`).

Dacă intenționați să rulați mai multe instanțe ale aceleiași versiuni de server, toate opțiunile specifice serverului care trebuie configurate vor trebui specificate la rulare.

Proceduri de pornire pentru servere multiple

Pornirea mai multor servere este mai complicată decât utilizarea unui singur server, deoarece `safe_mysqld` și `mysql.server` preferă ambele să lucreze într-o structură cu un singur server. Vă recomand să studiați cu atenție scriptul `safe_mysqld` și să-l folosiți ca bază pentru procedurile dumneavoastră de pornire, dar și să folosiți versiuni modificate pe care să le puteți adapta mai precis propriilor cerințe.

O problemă pe care trebuie să o rezolvați ține de specificarea opțiunilor în fișierele cu opțiuni. Cu servere multiple, nu puteți folosi fișierul `/etc/my.cnf` pentru parametri care variază de la un server la altul. Puteți folosi acel fișier numai pentru parametri care doriți să fie aceiași pentru toate serverele. Dacă fiecare server are o altă locație compilată intern a catalogului de date, puteți indica parametri specifici serverului în fișierul `my.cnf` din catalogul de date al fiecărui server. Cu alte cuvinte, folosiți `/etc/my.cnf` pentru parametri care doriți să fie utilizați de toate serverele, respectiv `DATADIR/my.cnf` pentru parametri specifici serverului, unde `DATADIR` variază de la un server la altul.

O altă modalitate de a specifica opțiunile pentru server este de a folosi `--defaults-file=nume_cale` ca primă opțiune în linia de comandă, pentru a indica serverului să citească opțiunile din fișierul specificat de `nume_cale`. Astfel, puteți insera opțiunile unui server într-un fișier unic pentru serverul respectiv, iar apoi îi cereți serverului să citească acel fișier la pornire. Rețineți că, dacă specificați această opțiune, nu va fi folosit nici unul dintre fișierele de opțiune uzuale (cum ar fi `/etc/my.cnf`).

Actualizarea sistemului MySQL

Când MySQL a fost lansat prima dată pentru public, se afla la versiunea 3.11.1. Acum, se află la versiunea seriei 3.22 pentru variantele stabile, respectiv 3.23 pentru variantele de dezvoltare. Numărul de versiune al seriei de versiuni stabile este întotdeauna cu o unitate minoră mai mic decât omologul său pentru seria de dezvoltare. După ce 3.23 va deveni stabilă, se va începe lucrul la seria de dezvoltare 3.24. Se pare că dezvoltatorii sistemului MySQL lucrează non-stop, deci variantele actualizate apar destul de frecvent (de mai multe ori într-un an). Sunt emise variante actualizate atât pentru seria stabilă, cât și pentru cea de dezvoltare. Ritmul actual de dezvoltare pune în fața administratorului MySQL problema de a actualiza sau nu versiunea instalată a sistemului la apariția de noi versiuni. Această secțiune conține unele îndrumări în sprijinul luării acestei decizii.

Primul lucru pe care îl aveți de făcut la apariția unei noi versiuni este de a afla care sunt diferențele față de variantele anterioare. Examinați anexa „Note de schimbare" din manualul de referință MySQL, pentru a vă familiariza cu diferențele. Apoi, puneți-vă următoarele întrebări:

- Aveți probleme cu versiunea mai veche, probleme rezolvate în noua versiune?
- Versiunea nouă dispune de caracteristici suplimentare pe care doriți sau aveți nevoie să le folosiți?
- Există creșteri de performanță pentru anumite tipuri de operații pe care le folosiți?

Dacă răspunsul la toate aceste întrebări este negativ, atunci nu aveți nici un motiv obligatoriu pentru a trece la o versiune mai recentă. Dacă răspunsul la oricare dintre aceste întrebări este afirmativ, atunci puteți continua. În acest moment, este deseori util să așteptați câteva zile și să examinați lista de corespondență pentru MySQL, pentru a vedea care este părerea celorlalți utilizatori despre noua versiune. Versiunea actualizată este utilă? Au fost descoperite hibe sau alte probleme?

Iată alți factori pe care îi puteți lua în considerare pentru a vă ajuta să luați o decizie:

- Variantele din seria stabilă au cel mai adesea rolul de a remedia unele hibe, nu de a include caracteristici noi. În general, riscul este mai redus în cazul variantelor actualizate din cadrul seriei stabile decât la variantele din seria de dezvoltare. (Desigur, dacă rulați un server cu seria de dezvoltare, pentru dumneavoastră riscul nu reprezintă o problemă atât de mare!)
- Este posibil ca, dacă actualizați sistemul MySQL, să fie necesar să actualizați și alte programe care sunt construite având legată biblioteca client C pentru MySQL. De exemplu, după o actualizare MySQL, poate fi necesară și reconstruirea programelor PHP, Apache și a modulului Perl DBD::mysql, pentru a lega noua bibliotecă client la aceste programe. (Un simptom evident care denotă necesitatea unei actualizări este faptul că toate scripturile dumneavoastră DBI și PHP legate de MySQL încep să transfere elemente de esență după actualizarea sistemului MySQL.) În general, reconstruirea acestor programe nu este o problemă chiar atât de dificilă, dar, dacă preferați să o evitați, ar fi mai bine să nu actualizați sistemul MySQL. Dacă folosiți programe legate static, nu dinamic, probabilitatea de apariție a acestei probleme este în mare măsură diminuată. Pe de altă parte, cerințele de memorie ale sistemului dumneavoastră vor crește.

În cazul în care nu sunteți încă sigur dacă să treceți sau nu la versiunea mai recentă, puteți testa întotdeauna noul server în mod independent de serverul dumneavoastră curent. Puteți face aceasta fie rulând noul server în paralel cu serverul de producție, fie instalându-l pe un alt calculator. Dacă folosiți un alt calculator, este mai ușor să mențineți independența între servere. Dacă nu vă permiteți luxul unui calculator separat pentru testare, puteți rula noul server pe calculatorul dumneavoastră de producție. Când procedați astfel, *nu uitați* să rulați noul server cu valori diferite pentru parametri precum locația de instalare, catalogul de date și portul de rețea și soclul la care serverul așteaptă conexiunile. Consultați secțiunea „Rularea mai multor servere" anterior în acest capitol.

Indiferent de caz, probabil veți dori să testați serverul folosind o copie a datelor din bazele dumneavoastră de date existente. Pentru instrucțiuni privind copierea bazelor de date, consultați secțiunea „Realizarea copiilor de siguranță și a copiilor obișnuite ale bazelor de date".

Dacă vă decideți să instalați o versiune actualizată, examinați anexa „Note de schimbare" din manualul de referință MySQL, pentru a vedea dacă există observații privind anumite operații speciale pe care trebuie să le efectuați în cursul procesului de actualizare. De regulă, asemenea mențiuni nu există, dar verificați, pentru orice eventualitate.

Nu vă temeți să încercați variantele de dezvoltare

Poate că preferați să nu folosiți variantele de dezvoltare cu bazele de date de producție, dar vă recomand măcar să încercați un server de test separat, eventual cu o copie a datelor de producție. Cu cât variantele noi sunt testate de mai multe persoane, cu atât detectarea hibelor este mai simplă. La anumite produse de baze de date, depistarea hibelor este o operație extrem de dificilă. Cu MySQL, rapoartele privind hibele constituie un factor semnificativ în evoluția procesului de dezvoltare a sistemului, deoarece dezvoltatorii remediază, de fapt, problemele pe care le semnalează comunitatea utilizatorilor.

CAPITOLUL 12

Securitate

Acest capitol se concentrează asupra acțiunilor pe care le poate întreprinde un administrator MySQL pentru menținerea securității și a integrității unui sistem MySQL. Deja am abordat câteva aspecte legate de securitate în capitolul 11, „Administrarea generală a sistemului MySQL", precum importanța stabilirii parolei inițiale a utilizatorului root din MySQL și configurarea conturilor de utilizator. Aceste subiecte au fost abordate ca parte a procesului de punere în funcțiune a sistemului. În acest capitol, vom examina mai îndeaproape alte aspecte legate de securitate:

- De ce securitatea este importantă și care sunt categoriile de atacuri împotriva cărora trebuie să vă protejați
- Riscurile care pot apărea din partea utilizatorilor gazdei serverului (securitate internă) și cum se pot preveni acestea
- Riscurile provenite din partea clienților care se conectează la server prin rețea (securitate externă) și metodele de prevenire a acestora

Administratorul MySQL este responsabil cu păstrarea în siguranță a conținutului bazelor de date, astfel încât înregistrările să fie accesibile numai celor care se bucură de autorizația corespunzătoare. Acest fapt implică atât securitatea internă, cât și pe cea externă.

Securitatea internă se referă la aspectele implicate la nivelul sistemului de fișiere – cu alte cuvinte, protecția catalogului de date MySQL împotriva atacurilor lansate de persoane care dispun de conturi pe calculatorul unde rulează serverul. Chiar dacă vă asigurați că tabelele de acordare care controlează accesul clienților prin rețea sunt configurate corect, acest lucru nu este de prea mare ajutor dacă permisiunile de fișier pentru conținutul catalogului de date sunt atât de lejere, încât fișierele care corespund tabelelor respective pot fi pur și simplu înlocuite!

Securitatea externă acoperă aspectele legate de clienții care se conectează din exterior, în speță protecția serverului MySQL împotriva atacurilor lansate prin conexiuni care sosesc prin rețea la server, solicitând accesul la conținutul bazelor de date. Doriți să configurați tabelele de acordare MySQL astfel încât acestea să nu permită accesul la bazele de date gestionate de server decât dacă solicitantul furnizează un nume de utilizator și o parolă corecte.

Acest capitol conține un ghid cu privire la elementele pe care trebuie să le cunoașteți și oferă instrucțiuni privind modalitatea de prevenire a accesului neautorizat atât la nivel intern, cât și la nivel extern.

Securitatea internă: securizarea accesului la catalogul de date

Serverul MySQL furnizează un sistem flexibil de privilegii, implementat cu ajutorul tabelelor de acordare din baza de date mysql. Puteți configura conținutul acestor tabele astfel încât să permiteți sau să interziceți clienților accesul la baza de date în orice mod doriți. Astfel, este asigurată securitatea împotriva accesului neautorizat din rețea la datele dumneavoastră. Totuși, stabilirea unor măsuri de securitate performante privind accesul din rețea la bazele de date nu vă va fi de nici un folos dacă alți utilizatori ai gazdei serverului au acces direct la conținutul catalogului de date. Dacă nu aveți certitudinea că sunteți unica persoană care deschide sesiunea de lucru la calculatorul unde rulează serverul MySQL, trebuie să vă preocupe posibilitatea ca alte persoane care folosesc același calculator să obțină acces la catalogul de date.

Iată ceea ce doriți să protejați:

- **Fișierele din baza de date.** Evident, doriți să păstrați confidențialitatea bazelor de date folosite de server. De regulă, și în mod îndreptățit, posesorii bazelor de date consideră conținutul acestora ca având un caracter privat. Chiar dacă nu procedează astfel, este la latitudinea lor să pună la dispoziția publicului conținutul unei baze de date, fără ca acesta din urmă să devină expus datorită securității îndoielnice a catalogului bazei de date.
- **Fișierele jurnal.** Jurnalul general și jurnalul de actualizare trebuie să fie păstrate în siguranță, deoarece conțin textele interogărilor. Aceasta este o problemă generală, în sensul că orice persoană cu acces la fișierele jurnal poate monitoriza tranzacțiile care se produc în cadrul bazelor de date.

  O problemă mai specifică de securitate referitoare la fișierele jurnal este aceea că interogările precum GRANT și SET PASSWORD sunt consemnate în jurnal. Fișierele de jurnal general și de actualizare conțin textul interogărilor importante, inclusiv parolele. (MySQL folosește criptarea parolelor, dar aceasta se aplică la stabilirea conexiunii, după ce parolele au fost deja configurate. Procesul de stabilire a unei parole implică o interogare precum GRANT, INSERT sau SET PASSWORD, iar asemenea interogări sunt consemnate în jurnal sub formă de text simplu.) Dacă un atacator are acces de citire la jurnale, acesta nu trebuie decât să ruleze comanda grep asupra jurnalului pentru a căuta cuvinte precum GRANT sau PASSWORD, descoperind astfel informații importante.

Evident, nu doriți ca alți utilizatori ai gazdei serverului să aibă acces de scriere la fișierele din catalogul de date, deoarece aceștia pot devasta fișierele de stare sau tabelele din bazele de date. Dar și accesul la citire este la fel de periculos. Dacă fișierele unui tabel pot fi citite, este o banalitate să furi fișierele și să determini chiar sistemul MySQL să afișeze conținutul tabelului, în format text simplu. Cum? Astfel:

1. Instalați propriul dumneavoastră server MySQL intrus pe gazda serverului, dar cu un port, soclu și catalog de date diferite de acelea folosite de către serverul oficial.
2. Rulați mysql_install_db pentru a vă inițializa catalogul de date. Astfel, obțineți accesul la serverul dumneavoastră ca utilizator MySQL root, deci aveți un control total asupra mecanismului de acces al serverului. De asemenea, această operație configurează o bază de date test.

3. Copiați fișierele corespunzătoare tabelului sau tabelelor pe care doriți să le furați în catalogul test situat în catalogul de date al serverului dumneavoastră.
4. Porniți serverul intrus. Abracadabra! Puteți avea acces la tabele după dorință. Instrucțiunea SHOW TABLES FROM test indică faptul că dispuneți de o copie a tabelelor furate, iar SELECT * afișează în totalitate conținutul oricăruia dintre tabele.
5. Dacă doriți să fiți într-adevăr „rău", deschideți permisiunile pentru conturile de utilizator anonim asupra serverului dumneavoastră, astfel încât orice persoană să se poată conecta la server, oriunde s-ar găsi aceasta, pentru a obține accesul la baza dumneavoastră de date test. Acum, ați adus literalmente la cunoștința întregii planete conținutul tabelelor furate.

Gândiți-vă puțin la asta, apoi inversați perspectiva. V-ar plăcea să vă facă cineva una ca asta? Categoric nu.

Puteți determina în ce măsură catalogul dumneavoastră de date conține fișiere sau cataloage nesigure executând comanda ls -l în catalogul de date. Căutați fișierele sau cataloagele care au activate permisiunile group sau other. Iată un listing parțial al unui catalog de date care nu este sigur, ca și unele din cataloagele de baze de date din cadrul acestuia:

```
% ls -l
total 10148
drwxrwxr-x   11   mysqladm   wheel      1024   May    8  12:20   .
drwxr-xr-x   22   root       wheel       512   May    8  13:31   ..
drwx------    2   mysqladm   mysqlgrp    512   Apr   16  15:57   menagerie
drwxrwxr-x    2   mysqladm   wheel       512   Jan   25  20:43   mysql
drwxrwxr-x    7   mysqladm   wheel        12   Aug   31   1998   sql-bench
drwxrwxr-x    2   mysqladm   wheel      1536   May    6  06:11   test
drwx------    2   mysqladm   mysqlgrp   1024   May    6  18:43   tmp
...
```

După cum se poate vedea, unele cataloage din baza de date au permisiuni adecvate, în timp ce altele nu. Situația din acest exemplu s-a creat în timp. Permisiunile mai puțin restrictive au fost create de servere mai vechi, care au fost mai puțin severe decât serverele mai recente în ceea ce privește configurarea permisiunilor. (Observați că datele cele mai recente se află în dreptul cataloagelor cu restricții mai mari, în speță menagerie și tmp.) Versiunile curente de MySQL se asigură ca aceste fișiere să poată fi citite numai de utilizatorul sub al cărui cont rulează serverul.

Să „reparăm" aceste permisiuni astfel încât numai utilizatorul serverului să aibă acces la cataloagele respective. Principalul dumneavoastră mijloc de protecție este dat de instrumentele furnizate de însuși sistemul de fișiere UNIX pentru stabilirea proprietarului și a modului fișierelor și cataloagelor. Iată ce aveți de făcut:

1. Treceți în catalogul de date:

   ```
   % cd DATADIR
   ```
2. Stabiliți ca toate fișierele aflate în catalogul de date să intre în posesia contului folosit pentru rularea serverului. (Trebuie să executați această operație din contul root.) În această carte, folosesc mysqladm și mysqlgrp ca nume de utilizator, respectiv de grup, ale acestui cont. Puteți modifica proprietarul folosind oricare dintre următoarele comenzi:

```
# chown -R mysqladm.mysqlgrp .
# find . -follow -type d -print | xargs chown mysqladm.mysqlgrp
```

3. Schimbați modul catalogului de date și al cataloagelor de baze de date, astfel încât acestea să poată fi citite numai de către mysqladm. Aceasta împiedică accesul altor utilizatori la conținutul catalogului dumneavoastră de date. Puteți proceda astfel folosind oricare dintre următoarele comenzi, care pot fi rulate fie ca root, fie ca mysqladm (este de preferat ultima situație, pentru a se reduce la minimum numărul de comenzi rulate din contul root):

```
% chmod -R go-rwx .
% find . -follow -type d -print | xargs chmod go-rwx
```

4. Proprietarul și modul conținutului catalogului de date sunt configurate pentru utilizatorul mysqladm. Acum, trebuie să vă asigurați că întotdeauna rulați serverul ca mysqladm, deoarece acesta este singurul utilizator care are acum acces la catalogul de date. Procedura pentru rularea serverului ca utilizator non-root este prezentată în capitolul 11.

După ce ați urmat instrucțiunile precedente, permisiunile asupra conținutului catalogului de date se modifică după cum urmează:

```
% ls -l
total 10148
drwxrwx---   11  mysqladm  wheel     1024  May   8 12:20  .
drwxr-xr-x   22  root      wheel      512  May   8 13:31  ..
drwx------    2  mysqladm  mysqlgrp   512  Apr  16 15:57  menagerie
drwx------    2  mysqladm  wheel      512  Jan  25 20:43  mysql
drwx------    7  mysqladm  wheel      512  Aug  31  1998  sql-bench
drwx------    2  mysqladm  wheel     1536  May   6 06:11  test
drwx------    2  mysqladm  mysqlgrp  1024  May   6 18:43  tmp
...
```

Securitatea externă: securizarea accesului prin rețea

Sistemul de securitate MySQL este flexibil, permițându-vă să configurați privilegiile de acces ale utilizatorilor în diferite moduri. În mod normal, configurarea privilegiilor de acces se execută cu ajutorul instrucțiunilor GRANT și REVOKE, ce modifică automat tabelele de acordare care controlează accesul clienților. Totuși, puteți dispune de o versiune mai veche de MySQL, care nu acceptă aceste instrucțiuni (instrucțiunile respective nu erau funcționale înainte de MySQL versiunea 3.22.11), sau puteți descoperi că privilegiile utilizatorilor nu funcționează așa cum doriți dumneavoastră. Pentru situații ca acestea, este util să înțelegeți structura tabelelor de acordare ale sistemului MySQL, precum și modul în care serverul le folosește pentru a determina permisiunile de acces. O asemenea înțelegere vă permite să adăugați, să eliminați sau să modificați privilegiile utilizatorilor prin modificarea directă a tabelelor de acordare. De asemenea, vă permite să diagnosticați problemele la nivel de privilegii atunci când examinați tabelele.

Voi presupune că ați citit secțiunea „Gestionarea conturilor de utilizator" în capitolul 1 și că înțelegeți modul de funcționare a instrucțiunilor GRANT și REVOKE. Aceste do instrucțiuni vă oferă o modalitate convenabilă de a configura conturile de utilizat MySQL și de a asocia privilegii la acestea, dar nu reprezintă decât o interfață. Adevăra acțiune se desfășoară la nivelul tabelelor de acordare MySQL.

Structura și conținutul tabelelor de acordare din MySQL

Accesul clienților care se conectează la server prin rețea la bazele de date MySQL es controlat prin conținutul tabelelor de acordare. Aceste tabele se găsesc în baza de dat mysql și sunt inițializate în cursul procesului de instalare inițială a sistemului MySQL p un calculator (așa cum este prezentat în Anexa A, „Obținerea și instalarea programelor de exemplu). Cele cinci tabele de acordare, și anume user, db, host, tables_priv columns_priv, sunt prezentate în tabelele 12.1 și 12.2.

Tabelul 12.1 Structura tabelelor de acordare user, db și host

user	db	host
Coloanele cu domeniul de valabilitate a accesului		
Host	Host	Host
User	Db	Db
Password	User	
Coloanele cu privilegii pentru baze de date/tabele		
Alter_priv	Alter_priv	Alter_priv
Create_priv	Create_priv	Create_priv
Delete_priv	Delete_priv	Delete_priv
Drop_priv	Drop_priv	Drop_priv
Index_priv	Index_priv	Index_priv
Insert_priv	Insert_priv	Insert_priv
References_priv	References_priv	References_priv
Select_priv	Select_priv	Select_priv
Update_priv	Update_priv	Update_priv
Coloanele cu privilegii administrative		
File_priv	Grant_priv	Grant_priv
Grant_priv		
Process_priv		
Reload_priv		
Shutdown_priv		

Tabelul 12.2 Structura tabelelor de acordare `tables_priv` şi `columns_priv`

tables_priv	columns_priv
Coloanele cu domeniul de valabilitate a accesului	
Host	Host
Db	Db
User	User
Table_name	Table_name
Column_name	
Coloană cu privilegii	
Table_priv	Column_priv

Conţinutul tabelelor de acordare este utilizat după cum urmează:

- `user`

  Tabelul `user` conţine utilizatorii care se pot conecta la server şi parolele lor; de asemenea, specifică privilegiile globale (de super-utilizator) de care dispun utilizatorii, *dacă* dispun. Toate privilegiile care sunt activate în intrările din tabelul `user` sunt privilegii globale şi se aplică tuturor bazelor de date. Dacă activaţi aici privilegiul `DELETE`, utilizatorul menţionat în intrare poate şterge înregistrări din orice tabel, de exemplu. Gândiţi-vă bine înainte de a proceda astfel.

  În general, cel mai indicat este ca toate privilegiile specificate în intrările din tabelul `user` să fie dezactivate, urmând ca acestea să fie specificate folosind celelalte tabele, care sunt mai restrictive. Excepţiile le constituie super-utilizatorii precum `root`, dar sunt destul de rare.

- `db`

  Tabelul `db` conţine lista bazelor de date şi utilizatorii care au privilegii pentru accesul la acestea. Un privilegiu specificat aici se aplică tuturor tabelelor dintr-o bază de date.

- `host`

  Tabelul `host` este folosit în combinaţie cu tabelul `db` pentru a controla privilegiile de acces la bazele de date pentru anumite gazde la un nivel mai fin decât cel posibil folosind numai tabelul `db`. Acest tabel nu este afectat de instrucţiunile `GRANT` şi `REVOKE`, deci este posibil să nu-l folosiţi niciodată.

- `tables_priv`

  Tabelul `tables_priv` specifică privilegiile la nivel de tabel. Un privilegiu specificat aici se aplică tuturor coloanelor dintr-un tabel.

- `columns_priv`

  Tabelul `columns_priv` specifică privilegiile la nivel de coloană. Un privilegiu specificat aici se aplică unei anumite coloane dintr-un tabel.

În secţiunea „Configurarea utilizatorilor fără instrucţiunea `GRANT`" vom discuta despre modalitatea în care instrucţiunea `GRANT` acţionează pentru a modifica aceste tabele, precum şi despre modul în care puteţi obţine aceleaşi rezultate prin modificarea directă a tabelelor de acordare.

Tabelele `tables_priv` şi `columns_priv` au fost introduse în MySQL 3.22.11 (în acelaşi timp cu instrucţiunea `GRANT`). Dacă dispuneţi de o versiune mai veche a sistemului MySQL, baza dumneavoastră de date `mysql` va dispune numai de tabelele `user`, `db` şi `host`. Dacă aţi trecut la versiunea 3.22.11 sau la o versiune ulterioară de la o versiune mai veche, dar nu dispuneţi de tabelele `tables_priv` şi `columns_priv`, rulaţi scriptul `mysql_fix_privileges_tables` pentru a le crea.

Nu există nici un tabel `rows_priv`, deoarece MySQL nu furnizează privilegii la nivel de înregistrare. De exemplu, nu puteţi limita un utilizator la acele rânduri dintr-un tabel care conţin o anumită valoare într-o coloană. Dacă aveţi nevoie de această funcţionalitate, trebuie să o furnizaţi folosind programarea aplicaţiilor. Dacă doriţi să furnizaţi o blocare consultativă la nivel de înregistrare, puteţi proceda astfel folosind funcţia `GET_LOCK()` descrisă în Anexa C, „Referinţă de operatori şi funcţii".

Tabelele de acordare conţin două categorii de coloane: coloanele domeniului de valabilitate, care determină aplicabilitatea unei intrări, respectiv coloanele de privilegii, care determină privilegiile acordate de o intrare. (Unele dintre tabelele de acordare mai conţin şi alte coloane, dar acest fapt nu ne interesează acum.)

Coloanele domeniului de valabilitate din tabelele de acordare

Coloanele domeniului de valabilitate din tabelele de acordare specifică aplicabilitatea intrărilor din tabel. Fiecare intrare din tabelele de acordare conţine coloanele `User` şi `Host`, pentru a indica faptul că intrarea se aplică atunci când utilizatorul dat se conectează de la gazda dată. (Tabelul `host` este o excepţie; este folosit într-un mod special, despre care nu vom discuta deocamdată.) Celelalte tabele conţin coloane suplimentare privind domeniul de valabilitate. De exemplu, tabelul `db` conţine o coloană `Db` pentru a indica baza de date căreia i se aplică intrarea. Similar, tabelele `tables_priv` şi `columns_priv` conţin câmpuri de determinare a domeniului de valabilitate, care reduc acest domeniu la un anumit tabel dintr-o bază de date, respectiv la o anumită coloană dintr-un tabel.

Coloanele de privilegii din tabelele de acordare

Tabelele de acordare mai conţin şi coloane de privilegii. Acestea indică privilegiile deţinute de utilizatorul care este specificat în coloanele de determinare a domeniului de valabilitate. Privilegiile acceptate de MySQL sunt prezentate în lista următoare. Lista foloseşte numele privilegiilor care sunt utilizate pentru instrucţiunea `GRANT`. În majoritatea cazurilor, între numele coloanelor de privilegii din tabelele `user`, `db` şi `host` şi numele privilegiilor discutate în legătură cu instrucţiunea `GRANT` există o asemănare evidentă. De exemplu, coloana `Select_priv` corespunde privilegiului `SELECT`.

Privilegii de bază de date şi de tabel

Privilegiile următoare se aplică operaţiilor cu baze de date şi tabele:

- `ALTER`

  Vă permite să folosiţi instrucţiunea `ALTER TABLE`. Acesta este, de fapt, numai un simplu privilegiu de nivel unu; trebuie să mai dispuneţi şi de alte privilegii, în funcţie de ceea ce doriţi să faceţi cu tabelul.

- `CREATE`

  Vă permite să creați baze de date și tabele. Nu vă permite să creați indexuri.
- `DELETE`

  Vă permite să eliminați din tabele înregistrările existente.
- `DROP`

  Vă permite să ștergeți baze de date și tabele. Nu vă permite să ștergeți indexuri.
- `INDEX`

  Vă permite să creați sau să ștergeți indexuri din tabele.
- `INSERT`

  Vă permite să înregistrați în tabele înregistrări noi.
- `REFERENCES`

  Acest privilegiu nu este folosit în momentul de față.
- `SELECT`

  Vă permite să regăsiți date din tabele folosind instrucțiuni `SELECT`. Acest privilegiu nu este necesar pentru instrucțiunile `SELECT` care nu implică tabele, cum ar fi `SELECT NOW()` sau `SELECT 4/2`.
- `UPDATE`

  Vă permite să modificați înregistrările existente în tabele.

Privilegii administrative

Următoarele privilegii se aplică operațiunilor administrative care controlează operarea serverului sau capacitatea unui utilizator de a acorda privilegii:

- `FILE`

  Vă permite să cereți serverului să citească sau să scrie fișiere pe gazda serverului. Acest privilegiu nu trebuie acordat fără un motiv valabil; este un privilegiu periculos, așa cum se va arăta în secțiunea „Riscuri de evitat în ceea ce privește tabelele de acordare". Serverul își ia anumite măsuri de precauție pentru a păstra utilizarea acestui privilegiu între anumite limite. Puteți citi numai fișiere pe care le poate citi toată lumea și care nu sunt considerate ca fiind protejate într-un fel sau altul. Un fișier pe care îl scrieți nu trebuie să existe deja. Astfel, nu puteți obliga serverul să suprascrie fișiere importante, precum `/etc/passwd`, sau fișiere dintr-o bază de date care aparține unei alte persoane. De exemplu, dacă această restricție nu ar fi fost aplicată, ați fi putut înlocui în totalitate conținutul tabelelor de acordare din baza de date `mysql`.

  Dacă acordați privilegiul `FILE`, asigurați-vă că nu rulați serverul în calitate de utilizator UNIX `root`, deoarece acesta poate crea noi fișiere oriunde în sistemul de fișiere. Dacă rulați serverul ca utilizator fără privilegii, serverul poate crea fișiere numai în cataloagele accesibile utilizatorului respectiv.
- `GRANT`

  Vă permite să acordați altor utilizatcri privilegiile de care dispuneți, inclusiv privilegiul `GRANT`.

- `PROCESS`

  Vă permite să vizualizați informații despre firele (procesele) care se execută la un moment dat în server, folosind instrucțiunea `SHOW PROCESSLIST` sau comanda `mysqladmin processlist`. De asemenea, acest privilegiu vă permite să suprimați firele cu instrucțiunea `KILL` sau cu ajutorul comenzii `mysqladmin kill`.

  Puteți întotdeauna să vedeți sau să suprimați propriile dumneavoastră fire. Privilegiul `PROCESS` vă oferă posibilitatea de a executa aceste operații cu orice fir.
- `RELOAD`

  Vă permite să efectuați o varietate de operații de administrare a serverului. Puteți emite instrucțiunea SQL `FLUSH`. De asemenea, puteți efectua comenzile `mysqladmin reload`, `refresh`, `flush-hosts`, `flush-privileges` și `flush-tables`.

  Acordarea acestui privilegiu nu prezintă pericole, chiar dacă privilegiul este de natură administrativă.
- `SHUTDOWN`

  Vă permite să opriți serverul cu instrucțiunea `mysqladmin shutdown`.

  În tabelele `user`, `db` și `host`, fiecare privilegiu este specificat ca o coloană separată. Toate aceste coloane sunt declarate cu tipul `ENUM("N","Y")`, deci valoarea prestabilită pentru fiecare privilegiu este `"N"` (dezactivat). Privilegiile din tabelele `tables_priv` și `columns_priv` sunt reprezentate de un `SET`, care permite specificarea privilegiilor în orice combinație, folosind o singură coloană. Aceste două tabele sunt mai recente decât celelalte trei, motiv pentru care folosesc cea mai recentă reprezentare. (Este posibil ca tabelele `user`, `db` și `host` să fie reorganizate în viitor, astfel încât și acestea să reprezinte privilegiile folosind coloane `SET`.)

Coloana `Table_priv` din tabelul `tables_priv` este definită astfel:

```
SET('Select','Insert','Update','Delete','Create','Drop',
   'Grant','References','Index','Alter')
```

Coloana `Column_priv` din tabelul `columns_priv` este definită astfel:

```
SET('Select','Insert','Update','References')
```

Numărul privilegiilor pentru coloane este mai mic decât acela al privilegiilor pentru tabele, deoarece la nivel de coloană se pot executa mai puține operații. De exemplu, puteți crea un tabel, dar nu puteți crea o coloană izolată.

Tabelul `user` conține coloane pentru anumite privilegii care nu se găsesc în nici unul din celelalte tabele de acordare: `File_priv`, `Process_priv`, `Reload_priv` și `Shutdown_priv`. Aceste privilegii se aplică operațiilor pe care le cereți serverului să le efectueze și care nu sunt asociate cu nici o bază de date sau tabel anume. De exemplu, nu are sens să permiteți unui utilizator să închidă serverul în funcție de baza de date curentă.

Controlul accesului clienților de către server

Există două etape de control al accesului clienților atunci când folosiți MySQL. Prima etapă se produce atunci când încercați să vă conectați la server. Serverul examinează tabelul `user`, pentru a vedea dacă poate găsi o intrare care corespunde numelui dum-

neavoastră, gazdei de la care vă conectaţi şi parolei pe care aţi furnizat-o. Dacă nu există nici o corespondenţă, nu vă puteţi conecta. Dacă există o corespondenţă, conexiunea este stabilită şi treceţi la a doua etapă. În cadrul acestei etape, pentru fiecare interogare pe care o emiteţi, serverul verifică tabelele de acordare pentru a vedea dacă aveţi sau nu suficiente privilegii pentru a executa interogarea. Cea de-a doua etapă continuă până când sesiunea dumneavoastră de lucru cu serverul se încheie.

Această secţiune descrie mai detaliat regulile pe care le foloseşte serverul MySQL pentru a stabili o corespondenţă între intrările din tabelele de acordare şi cererile de conexiune sosite din partea clienţilor, respectiv interogările. Aici sunt incluse tipurile de valori care pot exista în coloanele domeniului de valabilitate din tabelul de acordare, modalitatea de combinare a informaţiilor privind privilegiile din tabelele de acordare, precum şi ordinea în care sunt verificate intrările din tabel.

Conţinutul coloanelor domeniului de valabilitate

Unele dintre coloanele domeniului de valabilitate necesită valori literale, dar majoritatea permit caractere de înlocuire sau alte valori speciale.

- `Host`

  O valoare din coloana `Host` poate fi un nume de gazdă sau un număr IP. Valoarea `localhost` înseamnă gazda locală, dar va corespunde numai dacă vă conectaţi folosind efectiv o gazdă `localhost`, nu şi dacă vă conectaţi folosind numele gazdei. Să presupunem că numele gazdei locale este `pit-viper.snake.net` şi că există două intrări pentru dumneavoastră în tabelul `user`, una cu o valoare `localhost` în coloana `Host`, iar alta cu valoarea `pit-viper.snake.net` în aceeaşi coloană. Intrarea cu valoarea `localhost` va corespunde numai dacă vă conectaţi la `localhost`. Cealaltă intrare va corespunde numai dacă vă conectaţi la `pit-viper.snake.net`. Dacă doriţi ca utilizatorii să se poată conecta indiferent de gazdă, aveţi nevoie de două intrări în tabelul `user`.

  De asemenea, puteţi specifica valori în coloana `Host` folosind caractere de înlocuire. Caracterele model din SQL `%` şi `_` pot fi folosite şi au aceeaşi semnificaţie ca atunci când folosiţi operatorul `LIKE` într-o interogare. (Nu este permisă utilizarea modelelor `REGEXP`.) Caracterele model din SQL sunt valabile atât pentru nume, cât şi pentru numere IP. De exemplu, `%.wisc.edu` corespunde oricărei gazde din domeniul `wisc.edu`, iar `%.edu` corespunde oricărei gazde din orice instituţie de învăţământ. Similar, `192.168.%` corespunde oricărei gazde din subreţeaua de clasă B 192.168, în timp ce `192.168.3.%` corespunde oricărei gazde din subreţeaua de clasă C 192.168.3.

  Valoarea `%` corespunde tuturor gazdelor şi se poate folosi pentru a permite unui utilizator să se conecteze de oriunde. O valoare vidă în coloana `Host` este similară cu `%`. (Excepţie: în tabelul `db`, o valoare vidă în coloana `Host` are semnificaţia „pentru alte informaţii, examinaţi tabelul `host`".) Acest proces este descris în secţiunea „Verificarea accesului interogărilor".

  Începând de la MySQL 3.23, mai puteţi specifica numere IP cu o mască de reţea care indică numărul de biţi folosit pentru numărul reţelei. De exemplu, `192.168.128.0/17` specifică un număr de reţea pe 17 biţi şi corespunde oricărei gazde a cărei adresă IP conţine în primii 17 biţi secvenţa `192.168.128`.

- `User`

  Numele de utilizator trebuie să fie valori literale sau să fie vide. O valoare vidă cores punde oricărui utilizator. Caracterul `%` inserat într-o coloană `User` *nu* înseamnă valoar vidă; în schimb, corespunde unui utilizator cu un nume literal `%`, ceea ce probabil c nu este ceea ce doriţi.

  Când o cerere de conexiune este comparată cu tabelul `user` şi intrarea care corespund cererii conţine o valoare vidă în coloana `User`, clientul este asimilat cu un utilizato anonim.

- `Password`

  Valorile parolelor sunt fie vide, fie nevide, iar caracterele de înlocuire nu sunt permis O parolă vidă nu înseamnă că se poate utiliza orice parolă; înseamnă că utilizatorul n trebuie să specifice nici o parolă.

  Parolele sunt stocate ca valori criptate, nu ca text literal. Dacă stocaţi o valoare literal în coloana `Password`, utilizatorul va fi incapabil să se conecteze! Instrucţiunea `GRANT` comanda `mysqladmin password` criptează automat parola pentru dumneavoastră, da dacă folosiţi instrucţiuni precum `INSERT`, `REPLACE`, `UPDATE` sau `SET PASSWORD`, nu uita să specificaţi parola folosind sintaxa `PASSWORD("parola_noua")` nu pur şi simpl `"parola_noua"`.

- `Db`

  În tabelele `columns_priv` şi `tables_priv`, valorile din coloanele `Db` trebuie să reprezint nume literale ale bazelor de date; caracterele model şi valorile vide nu sunt permise. Î tabelele `db` şi `host`, valorile `Db` trebuie să fie specificate literal sau folosind caractere model din SQL `%` sau `_` pentru a specifica un caracter de înlocuire. O valoare `%` sau vid corespunde oricărei baze de date.

- `Table_name`, `Column_name`

  Valorile din aceste coloane trebuie să reprezinte nume literale ale tabelelor sau a coloanelor; modelele şi valorile vide nu sunt permise.

  Unele coloane ale domeniului de valabilitate sunt tratate de server ca sensibile la dif renţa între majuscule şi minuscule, altele nu. Regulile sunt prezentate în tabelul 12. Reţineţi mai ales că valorile `Table_name` sunt întotdeauna tratate ca sensibile la dif renţa între majuscule şi minuscule, chiar dacă tratamentul aplicat numelor de tabel i interogări depinde de sensibilitatea la diferenţa între majuscule şi minuscule a si temului de fişiere pe care rulează serverul (sensibil la diferenţa între majuscule minuscule sub UNIX, dar insensibil sub Windows).

Tabelul 12.3 Sensibilitatea la diferenţa între majuscule şi minuscule în coloanele dome niului de valabilitate din tabelele de acordare

Coloană	Sensibilă la diferenţa între majuscule şi minuscule
`Host`	Nu
`User`	Da
`Password`	Da

Continua:

Tabelul 12.3 Continuare

Coloană	Sensibilă la diferența între majuscule și minuscule
Db	Da
Table_name	Da
Column_name	Nu

Verificarea accesului interogărilor

De fiecare dată când emiteți o interogare, serverul verifică dacă aveți sau nu suficiente privilegii pentru a o executa. Această verificare se efectuează prin examinarea, în această ordine, a tabelelor user, db, tables_priv și columns_priv, până când serverul ajunge la concluzia că beneficiați de acces sau până când a terminat de căutat, fără rezultate, în toate tabelele. Mai concret:

1. Serverul caută în intrarea din tabelul user care a corespuns când v-ați conectat inițial, pentru a vedea care sunt privilegiile globale pe care le aveți. Dacă aveți asemenea privilegii și ele sunt suficiente, serverul execută interogarea.
2. Dacă privilegiile dumneavoastră globale sunt insuficiente, serverul caută intrarea corespunzătoare dumneavoastră în tabelul db și adaugă privilegiile din intrarea respectivă la privilegiile dumneavoastră globale. Dacă rezultatul este suficient pentru interogare, serverul o execută.

Modul de stocare a parolelor în tabelul user

Parolele sunt afișate în tabelul user sub formă de șiruri criptate, deci nu vă puteți da seama care este parola unui utilizator chiar dacă aveți acces de citire la acest tabel. Pare să fie ceva de la sine înțeles că funcția PASSWORD() execută același tip de criptare ca și cea folosită pentru parolele UNIX; totuși, nu este așa.

Cele două tipuri de criptare sunt asemănătoare în sensul că ambele sunt într-un singur sens și că nu sunt reversibile, dar MySQL nu folosește același algoritm de criptare ca și UNIX. Aceasta înseamnă că, chiar dacă vă folosiți parola UNIX ca parolă MySQL, nu trebuie să vă așteptați ca șirurile criptate ale celor două parole să corespundă. Dacă doriți să folosiți criptarea UNIX pentru o aplicație, folosiți funcția CRYPT() în locul funcției PASSWORD().

3. În cazul în care combinația dintre privilegiile dumneavoastră globale și cele de la nivelul bazei de date este insuficientă, serverul continuă să caute, mai întâi în tabelul tables_priv și apoi în tabelul columns_priv.
4. Dacă, după verificarea tuturor tabelelor, constată că nu aveți permisiunile adecvate, serverul respinge încercarea dumneavoastră de a executa interogarea.

În termeni booleeni, privilegiile din tabelele de acordare sunt folosite de server după cum urmează:

```
user OR db OR tables_priv OR columns_priv
```

Văd că vă întrebați de ce descrierea precedentă se referă numai la patru tabele de acordare, când numărul lor este de cinci. Bun, m-ați prins... Modul real în care serverul verifică permisiunile de acces este acesta:

```
user OR (db AND host) OR tables_priv OR columns_priv
```


Am prezentat mai întâi expresia mai simplă, deoarece tabelul host nu este afectat de instrucțiunile GRANT sau REVOKE. Dacă folosiți întotdeauna GRANT și REVOKE pentru a gestiona privilegiile utilizatorilor, nu vă interesează tabelul host. Dar iată care este procedeul, dacă vreți să știți:

- Când serverul caută privilegiile la nivel de bază de date, caută clientul în intrarea din tabelul db. Dacă valoarea din coloana Host este vidă, aceasta înseamnă „Caută în tabelul host pentru a afla care sunt gazdele ce pot obține accesul la baza de date".
- Serverul caută în tabelul host intrările care au în coloana Db aceeași valoare ca și intrarea din tabelul db. Dacă nici o intrare din tabelul host nu corespunde gazdei clientului, nu va fi acordat nici un privilegiu la nivel de baze de date. Dacă vreuna din aceste intrări are în coloana Host o valoare care corespunde gazdei de la care se conectează clientul, intrarea din tabelul db și intrarea din tabelul host sunt combinate, generând privilegiile la nivel de bază de date ale clientului.

  Totuși, privilegiile sunt combinate folosind un AND (ȘI) logic, ceea ce înseamnă că clientul nu are un privilegiu dat decât dacă acesta se află în ambele intrări. În acest mod, puteți acorda un set elementar de privilegii în intrarea din tabelul db, iar apoi le dezactivați în mod selectiv pentru anumite gazde, folosind intrările din tabelul host. De exemplu, puteți permite accesul la o bază de date de la toate gazdele din domeniul dumneavoastră, dar dezactivați privilegiile la nivel de bază de date pentru gazdele situate în zone mai puțin sigure.

Indiscutabil, descrierea anterioară face procesul de verificare a accesului să pară destul de complicat, mai ales dacă ne gândim că serverul verifică privilegiile pentru fiecare interogare emisă de clienți. Totuși, procesul este foarte rapid, deoarece serverul nu caută informații din tabelele de acordare pentru fiecare interogare în parte. În schimb, la pornire serverul citește conținutul tabelelor în memorie, apoi verifică interogările folosind copiile din memorie. Astfel, performanțele operațiilor de verificare a accesului cresc considerabil, dar se înregistrează și un efect marginal important: dacă modificați direct conținutul tabelelor de acordare, serverul nu va remarca modificarea de privilegii.

De exemplu, dacă adăugați un nou utilizator MySQL utilizând o instrucțiune INSERT pentru a insera o nouă înregistrare în tabelul user, utilizatorul denumit în cadrul intrării nu se va putea conecta la server. Acest fapt îi derutează deseori pe administratorii începători (uneori chiar și pe cei mai experimentați!), dar soluția este foarte simplă: indicați serverului să reîncarce conținutul tabelelor de acordare, după ce le-ați modificat. Puteți proceda astfel emițând o instrucțiune FLUSH PRIVILEGES sau executând comanda mysqladmin flush-privileges (sau mysqladmin reload, dacă aveți o versiune veche, care nu acceptă flush-privileges).

Ordinea de corespondență a coloanelor domeniului de vizibilitate

Serverul MySQL stochează intrările în tabelele de acordare într-un anumit mod, apoi încearcă să stabilească corespondențele pentru solicitările de conexiune examinând intrările în ordine. Prima corespondență găsită determină intrarea care va fi folosită. Este important să înțelegeți ordinea de sortare folosită de MySQL, mai ales pentru tabelul user. Acest fapt pare să deruteze pe mulți care încearcă să înțeleagă securitatea sistemului MySQL.

Când serverul citeşte conţinutul tabelului user, sortează intrările în funcţie de valorile din coloanele Host şi User. Valoarea Host este valoare dominantă (intrările cu aceeaşi valoare Host sunt sortate împreună, apoi ordonate în funcţie de valoarea din coloana User). Totuşi, sortarea *nu* se execută în ordine lexicografică, ci numai parţial. Principiul care trebuie reţinut este acela că valorile literale sunt preferate în detrimentul modelelor. Aceasta înseamnă că, dacă vă conectaţi de la boa.snake.net şi există intrări cu valorile boa.snake.net şi %.snake.net în coloana Host, prima intrare va fi preferată. Similar, %.snake.net este preferată faţă de %.net, care la rândul său este preferată faţă de %. Stabilirea corespondenţelor pentru numerele IP funcţionează în acelaşi mod. Pentru un client care se conectează de la o gazdă cu un număr IP 192.168.3.14, vor corespunde intrările 192.168.3.14, 192.168.3.%, 192.168.%, 192.% şi %, în această ordine.

Riscuri de evitat în ceea ce priveşte tabelele de acordare

Această secţiune descrie unele măsuri de precauţie care trebuie luate în considerare atunci când acordaţi privilegii, precum şi riscurile care însoţesc opţiunile nechibzuite. În general, este de dorit ca privilegiile de super-utilizator să fie acordate cu economie. Cu alte cuvinte, nu activaţi privilegiile din intrările tabelului user. Utilizaţi în schimb celelalte tabele de acordare, pentru a limita privilegiile utilizatorilor la anumite baze de date, tabele sau coloane. Privilegiile din tabelul user permit utilizatorului să influenţeze operarea serverului dumneavoastră, respectiv să obţină accesul la orice tabel din orice bază de date.

Nu acordaţi privilegii pentru baza de date mysql. Un utilizator cu privilegii asupra bazei de date care conţine tabelele de acordare poate fi capabil să modifice tabelele pentru a dobândi privilegii şi asupra oricărei alte baze de date. Prin acordarea privilegiilor care permit unui utilizator să modifice tabelele din baza de date mysql, utilizatorul respectiv dobândeşte efectiv un privilegiu GRANT global. Dacă utilizatorul poate modifica tabelele în mod direct, acest lucru este aproape echivalent cu posibilitatea de a emite orice instrucţiune GRANT care îi trece prin cap respectivului.

Privilegiul FILE este deosebit de periculos; nu-l acordaţi cu uşurinţă. Iată un exemplu de acţiune pe care o poate executa un utilizator cu privilegiul FILE:

```
CREATE TABLE etc_parole (pwd_entry TEXT);
LOAD DATA INFILE "/etc/passwd" INTO TABLE etc_parole;
SELECT * FROM etc_parole;
```

După ce a emis aceste instrucţiuni, utilizatorul dispune de conţinutul fişierului dumneavoastră cu parole. De fapt, conţinutul *oricărui* fişier care poate fi citit public din server este accesibil prin reţea de către orice utilizator cu privilegiul FILE.

De asemenea, privilegiul FILE poate fi exploatat pentru a compromite bazele de date pe sisteme care nu sunt configurate cu permisiuni de fişier suficient de restrictive. Acesta este un motiv pentru care trebuie să decideţi ca numai serverul să poată citi conţinutul catalogului de date. Dacă fişierele corespunzătoare tabelelor din bazele de date pot fi citite de oricine, atunci pot fi citite nu numai de către orice utilizator cu un cont pe gazda serverului, dar şi de orice utilizator al programelor client cu privilegiul FILE care se poate conecta prin reţea! Următoarea procedură demonstrează cum se poate lucra:

- Creaţi un tabel cu o coloană LONGBLOB:

```
USE test
CREATE TABLE tmp (b LONGBLOB)
```

- Folosiţi tabelul pentru a citi în acesta conţinutul fiecărui fişier corespunzător tabelului pe care doriţi să-l furaţi, după care scrieţi conţinutul tabelului într-un fişier din propria dumneavoastră bază de date:

```
LOAD DATA INFILE "./other_db/x.frm" INTO TABLE tmp
   FIELDS ESCAPED BY "" LINES TERMINATED BY ""
SELECT * FROM tmp INTO OUTFILE "y.frm"
   FIELDS ESCAPED BY "" LINES TERMINATED BY ""
DELETE FROM tmp
LOAD DATA INFILE "/.other_db/x.ISD" INTO TABLE tmp
   FIELDS ESCAPED BY "" LINES TERMINATED BY ""
SELECT * FROM tmp INTO OUTFILE "y.ISD"
   FIELDS ESCAPED BY "" LINES TERMINATED BY ""
DELETE FROM tmp
LOAD DATA INFILE "/.other_db/x.ISM" INTO TABLE tmp
   FIELDS ESCAPED BY "" LINES TERMINATED BY ""
SELECT * FROM tmp INTO OUTFILE "y.ISM"
   FIELDS ESCAPED BY "" LINES TERMINATED BY ""
```

- În acest moment, aveţi un nou tabel y, care include conţinutul tabelului other_db.x, iar dumneavoastră aveţi acces total la acesta.

Pentru a evita să fiţi atacat în acest mod, stabiliţi permisiunile asupra conţinutului catalogului dumneavoastră de date în conformitate cu instrucţiunile din secţiunea „Securitate internă: securizarea accesului la catalogul de date". De asemenea, puteţi folosi opţiunea --skip-show-database atunci când porniţi serverul, pentru a limita folosirea instrucţiunilor SHOW DATABASES şi SHOW TABLES de către utilizatori pentru baze de date la care aceştia nu au acces. Astfel, utilizatorii nu mai pot afla despre existenţa unor baze de date, respectiv tabele, la care nu trebuie să aibă acces.

Privilegiul ALTER poate fi folosit în moduri pe care nu le imaginaţi. Să presupunem că dorim ca utilizatorul user1 să poată obţine accesul la tabel1, dar nu şi la tabel2. Un utilizator cu privilegiul ALTER vă poate deturna intenţia folosind ALTER TABLE pentru a redenumi tabel2 cu numele tabel1.

Fiţi atent la privilegiul GRANT. Doi utilizatori cu privilegii diferite, care dispun ambii de privilegiul GRANT, îşi pot mări reciproc puterea drepturilor de acces ale fiecăruia.

Configurarea utilizatorilor fără instrucţiunea GRANT

Dacă dispuneţi de o versiune a sistemului MySQL anterioară versiunii 3.22.11, nu puteţi folosi instrucţiunile GRANT sau REVOKE pentru a configura utilizatori şi privilegii de acces, dar puteţi aduce modificări în mod direct conţinutului tabelelor de acordare. O bună parte din aceste operaţii sunt mai simple dacă înţelegeţi cum acţionează instrucţiunea GRANT pentru a modifica tabelele de acordare. Apoi, puteţi proceda în acelaşi mod prin

emiterea manuală a instrucțiunilor `INSERT`. (Instrucțiunea `INSERT` poate fi dizgrațioasă și dificil de introdus corect, dar aceasta este o altă problemă. „Urâțenia" instrucțiunii `INSERT` este, de fapt, unul dintre motivele pentru care instrucțiunea `GRANT` este cu mult mai simplu de utilizat.)

Când emiteți o instrucțiune `GRANT`, specificați un nume de utilizator și un nume de gazdă, eventual o parolă. Pentru utilizator este creată o intrare în tabelul `user`, iar aceste valori sunt înregistrate în coloanele `User`, `Host` și `Password` ale intrării. Dacă specificați privilegii globale în instrucțiunea `GRANT`, aceste privilegii sunt înregistrate în coloanele cu privilegii ale intrării. Trebuie să fiți atent la faptul că instrucțiunea `GRANT` criptează parola automat, dar `INSERT` nu execută această operație; va trebui să folosiți funcția `PASSWORD()` pentru a cripta parolele în instrucțiunile dumneavoastră `INSERT`.

Dacă ați specificat privilegii la nivel de bază de date, numele de utilizator și numele gazdei sunt înregistrate în coloanele `User` și `Host` ale unei intrări din tabelul `db`. Baza de date pentru care ați acordat privilegii este înregistrată în coloana `Db`, iar privilegiile pe care le-ați acordat sunt înregistrate în coloanele pentru privilegii.

Pentru privilegiile la nivel de tabel, respectiv la nivel de coloană, efectele sunt asemănătoare. Sunt create intrări în tabelele `tables_priv` și `columns_priv` pentru a înregistra numele de utilizator, numele gazdei și baza de date, precum și tabelul, respectiv tabelul și coloana, dacă este necesar. Privilegiile acordate sunt înregistrate în coloana cu privilegii.

Dacă ați reținut descrierea anterioară, trebuie să puteți executa operațiile efectuate de instrucțiunea `GRANT` chiar în absența acesteia. Rețineți că, atunci când modificați în mod direct tabelele de acordare, va trebui să indicați serverului să le reîncarce, deoarece în caz contrar acesta nu va sesiza modificările. Puteți forța o reîncărcare executând o comandă `mysqladmin flush-privileges` sau `mysqladmin reload`. Dacă omiteți acest lucru, vă veți întreba de ce serverul nu se comportă așa cum doriți.

Următoarea instrucțiune `GRANT` creează un super-utilizator dotat cu toate privilegiile, inclusiv capacitatea de a acorda privilegii altor utilizatori:

```
GRANT ALL ON *.* TO elena@localhost IDENTIFIED BY "cafea"
    WITH GRANT OPTION
```

Instrucțiunea va crea o intrare pentru `elena@localhost` în tabelul `user` și va activa toate privilegiile din tabelul respectiv, deoarece acolo sunt stocate privilegiile de super-utilizator (globale). Pentru a face același lucru cu `INSERT`, instrucțiunea este:

```
INSERT INTO user VALUES ("localhost","elena",PASSWORD("cafea")),
    "Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y","Y")
```

Asta este o instrucțiune `INSERT` urâtă de-a binelea! Puteți chiar descoperi că nu funcționează, în funcție de versiunea dumneavoastră a sistemului MySQL. Poate că structura tabelelor de acordare s-a modificat cu o anumită ocazie și s-ar putea să nu aveți 14 coloane cu privilegii în tabelul dumneavoastră `user`. Folosiți `SHOW COLUMNS` pentru a afla care sunt coloanele cu privilegii pe care le conține fiecare tabel de acordare și modificați în consecință instrucțiunile dumneavoastră `INSERT`.

Următoarea instrucțiune `GRANT` mai creează un utilizator cu statut de super-utilizator, dar numai pentru un singur privilegiu:

```
GRANT RELOAD ON *.* flush@localhost IDENTIFIED BY "flushpass"
```


Poate vă reamintiți de această instrucțiune din capitolul 11, când am creat utilizatorul `flush`. Instrucțiunea `INSERT` pentru acest exemplu este ceva mai simplă decât pentru exemplul anterior; este mai simplu să se menționeze numele coloanelor și să se specifice singura coloană cu privilegii. Toate celelalte coloane vor primi în mod prestabilit valoarea `"N"`:

```
INSERT INTO user (Host,User,Password,Reload_priv)
    VALUES("localhost","flush",PASSWORD("flushpass"),"Y")
```

Privilegiile la nivel de bază de date sunt acordate folosind o clauză `ON nume_baza_de_date.*`, în loc de `ON *.*`:

```
GRANT ALL ON samp_db.* TO boris@localhost IDENTIFIED BY "rubin"
```

Aceste privilegii nu sunt globale, deci nu vor fi stocate în tabelul `user`. Trebuie să creăm o intrare în tabelul `user` (astfel încât utilizatorul să se poată conecta), dar trebuie să mai creăm o intrare și în tabelul `db`, pentru a înregistra privilegiile la nivel de bază de date:

```
INSERT INTO user (Host,User,Password)
    VALUES("localhost","boris",PASSWORD("rubin"))
INSERT INTO db VALUES("localhost","samp_db","boris",
    "Y","Y","Y","Y","Y","Y","N","Y","Y","Y")
```

Coloana cu valoarea `"N"` este cea aferentă privilegiului `GRANT`; pentru o instrucțiune `GRANT` la nivel de bază de date însoțită de opțiunea `WITH GRANT OPTION`, valoarea din acea coloană ar fi fost `"Y"`.

Pentru a configura privilegii la nivel de tabel sau la nivel de coloană, folosiți instrucțiuni `INSERT` pentru tabelele `tables_priv` sau `columns_priv`. Desigur, dacă nu dispuneți de instrucțiunea `GRANT`, nu veți beneficia nici de aceste tabele, deoarece toate au apărut în MySQL în același timp. Dacă aveți tabelele și doriți să le manipulați manual dintr-un anumit motiv, rețineți că nu se activează privilegiile folosind coloane individuale. Pentru aceasta configurați fie coloana `tables_priv.Table_priv`, fie coloana `columns_priv.Column_priv` la o valoare `SET` alcătuită din privilegiile pe care doriți să le activați. De exemplu, pentru a activa privilegiile `INSERT` și `SELECT` pentru un tabel, veți atribui în coloana `Table_priv` o valoare `"Select,Insert"` în intrarea relevantă din tabelul `tables_priv`.

Dacă doriți să modificați privilegiile pentru un utilizator al cărui cont MySQL există deja, folosiți `UPDATE` în loc de `INSERT`. Acest fapt este valabil indiferent dacă adăugați sau revocați privilegii. Pentru a elimina complet un utilizator, folosiți `DELETE` pentru a elimina intrările aferente utilizatorului din fiecare tabel de acordare.

Dacă preferați să nu emiteți interogări pentru a modifica tabelele de acordare în mod direct, puteți examina scripturile `mysqlaccess` și `mysql_setpermissions` care sunt incluse în distribuția MySQL.

O enigmă cu privilegii (Partea a II-a)

În capitolul 11, în secțiunea „O enigmă cu privilegii (Partea I)“, am examinat o situație de acordare a privilegiilor în care privilegiile nu aveau efectul scontat. Pentru a ne reaminti rapid contextul respectiv, o problemă care se produce frecvent cu instalările noi ale sistemului MySQL este aceea că administratorul dorește să adauge o intrare pentru un utilizator, intrare care să-i permită acestuia să se conecteze de la mai multe gazde. Modalitatea evidentă de a proceda constă în utilizarea unui specificator al numelui gazdei care conține caracterul de înlocuire `%`, astfel încât administratorul creează un utilizator folosind o instrucțiune care seamănă cu aceasta:

```
GRANT ALL ON samp_db.* TO fred@%.snake.net IDENTIFIED BY "cacao"
```

Se întâmplă ca `fred` să aibă un cont pe gazda serverului, deci el încearcă să se conecteze de acolo:

```
% mysql -u fred -pcacao samp_db
ERROR 1045: Access denied for user: 'fred@localhost'
(Using password: YES)
```

De ce? Pentru a înțelege ce se întâmplă, trebuie să examinăm atât modul în care `mysql_install_db` configurează tabelele de acordare inițiale, cât și modul în care serverul folosește intrările din tabelul `user` pentru a stabili echivalențe cu parametrii conexiunilor client. Când vă inițializați bazele de date prin rularea programului `mysql_install_db`, acesta creează intrări în tabelul `user` cu valori în coloana `User` și `Host` astfel:

Host	User
localhost	root
pit-viper.snake.net	root
localhost	
pit-viper.snake.netroot	

Primele două intrări permit utilizatorului `root` să se conecteze la gazda locală specificând fie `localhost`, fie numele gazdei. Celelalte două permit stabilirea unor conexiuni anonime de la gazda locală. La adăugarea intrării pentru utilizatorul `fred`, tabelul va conține următoarele intrări:

Host	User
localhost	root
pit-viper.snake.net	root
localhost	
pit-viper.snake.netroot	
%.snake.netroot	fred

Când serverul pornește, citește intrările și le sortează (mai întâi în funcție de gazdă, apoi în funcție de utilizatorul gazdei), începând de la valorile mai specifice și terminând cu cele mai generale:

Host	User
localhost	root
localhost	
pit-viper.snake.net	root
pit-viper.snake.net	
%.snake.netroot	fred

Cele două intrări cu `localhost` sunt sortate alături, iar apoi intrarea pentru `root` est inserată prima, deoarece este mai specifică decât valoarea vidă. Intrările cu `pit-viper.snake.net` sunt sortate alături într-un mod similar. Toate aceste intrări au valor literale în coloana `Host`, fără caractere de înlocuire, deci toate vor fi sortate anterio intrării pentru utilizatorul `fred`, care folosește un caracter de înlocuire. În particula intrările pentru utilizatori anonimi au prioritate față de intrarea utilizatorului `fred` î ordinea de sortare.

Rezultatul este acela că, atunci când `fred` încearcă să se conecteze de la gazda locală, un dintre intrările cu nume de utilizator vid este echivalentă cu parametrii de conexiun anterior intrării care conține valoarea `%.snake.net` în coloana `Host`. Parola aferent intrării respective este vidă, deoarece utilizatorii anonimi prestabiliți nu au parol Deoarece `fred` a specificat o parolă atunci când s-a conectat, a apărut o neconcordanț între parole și conexiunea a eșuat.

Aici trebuie reținut că, deși este foarte convenabilă utilizarea caracterelor de înlocuir pentru a specifica gazdele de la care se pot conecta utilizatorii, puteți avea probleme conexiunile de la gazda locală, atâta timp cât lăsați în tabelul `user` intrările aferente ut lizatorilor anonimi.

În general, vă recomand să ștergeți intrările pentru utilizatorii anonimi. Veți avea viață mai ușoară:

```
mysql DELETE FROM user WHERE User="";
```

Dacă doriți să fiți mai concret, ștergeți toate intrările aferente utilizatorilor anonimi di celelalte tabele de acordare. Tabelele care conțin coloane `user` sunt `db`, `tables_priv` `columns_priv`.

Enigma prezentată în această secțiune a abordat o anumită situație, dar conține o lecț cu un caracter mai general. Dacă privilegiile aferente unui utilizator nu funcționează a cum este de dorit, examinați tabelele de acordare, pentru a vedea dacă există vreo intra care să conțină în coloana `Host` valori care au un caracter mai specific decât intrarea al care să conțină în coloana Host valori care vor corespunde parametrilor conexiunii solicita către utilizatorului în chestiune și care vor corespunde parametrilor conexiunii solicita de acel utilizator. Dacă aceasta este situația, ați găsit explicația problemei. Poate că es necesar să fiți mai concret în ceea ce privește intrarea aferentă utilizatorului (sau să adă gați o altă intrare, pentru a acoperi situația cu un caracter mai particular).

CAPITOLUL 13

Întreținerea și repararea bazelor de date

Într-un model ideal, MySQL rulează fără probleme începând cu momentul în care îl instalați. Dar uneori se produc și probleme, dintr-o varietate de motive, de la căderi de tensiune și defecțiuni ale componentelor hardware și până la o închidere în condiții inadecvate a serverului MySQL (cum ar fi oprirea execuției acestuia folosind comanda `kill -9` sau când calculatorul suferă o cădere). Circumstanțe precum acestea, din care multe depășesc posibilitatea dumneavoastră de control, pot avea ca rezultat deteriorări ale tabelelor din bazele de date, cauzate în mod caracteristic de scrieri incomplete care se produc în cursul modificării unui tabel.

Acest capitol se concentrează asupra detectării și rezolvării problemelor care pot apărea la tabelele dumneavoastră, indiferent de modul în care se produc aceste probleme. Pentru verificarea și repararea tabelelor, cei mai buni prieteni ai administratorului MySQL sunt utilitarele `myisamchk` și `isamchk`. Aceste două programe dispun de numeroase caracteristici. Deja am discutat despre modul de utilizare a acestora pentru efectuarea analizei distribuției cheilor de index, precum și pentru dezactivarea și reactivarea indexului în capitolul 4, „Optimizarea interogărilor". De asemenea, le puteți folosi pentru verificarea tabelelor și repararea acelora care prezintă probleme, fapt care vă permite să remediați pagubele înainte ca acestea să se agraveze, ducând chiar la imposibilitatea de utilizare a tabelelor.

Pentru o listă completă a opțiunilor acceptate de `myisamchk` și `isamchk`, consultați Anexa E, „Referință de programe MySQL". Pentru informații suplimentare, consultați capitolul „Întreținerea unei instalări a sistemului MySQL" din manualul de referință MySQL.

Verificarea și remedierea tabelelor din bazele de date

Procedura generală pentru detectarea pagubelor aduse unui tabel și pentru remedierea acestora este următoarea:

1. Verificați tabelul pentru a detecta eventualele erori.

   Dacă tabelul nu prezintă erori, ați terminat. În caz contrar, trebuie să-l reparați.
2. Executați copii ale fișierelor din tabel înainte de a începe reparațiile, pentru orice eventualitate.
3. Încercați să reparați tabelul.
4. Dacă operația de reparare eșuează, restaurați tabelul pornind de la copiile de siguranță ale bazei de date și de la jurnalele de actualizare.

Etapa finală a procedurii prezentate anterior presupune că ați efectuat copii de siguranță ale bazei de date și că ați activat caracteristica de consemnare în jurnalele de actualizare. Dacă nu este așa, atunci duceți o existență periculoasă. Citiți capitolul 11, „Administrarea generală a sistemului MySQL", pentru a afla modul în care puteți folosi `mysqldump` și modul de activare a consemnării în jurnal. Nu doriți să vă aflați vreodată în situația de a pierde un tabel fără posibilitatea de a-l recupera, fiindcă v-a fost lene să executați copii de siguranță.

Înainte de a folosi `myisamchk` sau `isamchk` pentru a verifica sau repara tabele, este necesară satisfacerea unor cerințe preliminarii:

- Instituiți proceduri de executare sistematică a copiilor de siguranță a bazelor de date și activați consemnarea în jurnalele de actualizare, în cazul în care răul devine și mai rău și un tabel este atât de deteriorat, încât nu mai poate fi reparat. N-am mai spus asta?
- Parcurgeți în totalitate această secțiune înainte de a încerca vreuna dintre procedurile descrise. În particular, *nu* trebuie să folosiți instrucțiunile prezentate aici înainte de a fi citit secțiunea „Evitarea interacțiunilor cu serverul MySQL". Secțiunea respectivă tratează despre problemele care apar în cazul în care serverul folosește un tabel în același timp în care dumneavoastră încercați să-i aplicați tabelului proceduri de verificare sau reparare. De asemenea, secțiunea descrie modul de a preveni apariția acestor probleme prin păstrarea serverului în stare de repaus în timp ce lucrați.
- Când efectuați verificarea sau repararea tabelelor, trebuie să fiți conectat sub contul folosit pentru a rula `mysqld`, deoarece aveți nevoie de acces de tip citire și scriere la fișierele tabel.

Sintaxa de invocare a utilitarelor `myisamchk` și `isamchk`

Utilitarele MySQL `myisamchk` și `isamchk` sunt foarte asemănătoare și se utilizează în același mod în majoritatea cazurilor. Principala diferență dintre acestea constă în tipul de tabele pentru care este folosit fiecare utilitar. Pentru tabelele MyISAM se folosește `myisamchk`, iar pentru tabelele ISAM se folosește `isamchk`. Se poate determina tipul formatului de stocare folosit de fiecare tabel în funcție de extensia numelui de fișier al fișierului index al tabelului. Extensia `".MYI"` indică un tabel MyISAM, în timp ce extensia `".ISM"` indică un tabel ISAM.

Pentru a folosi oricare din aceste utilitare, indicați-i acestuia tabelele pe care doriți să le verificați sau să le reparați, alături de opțiunile care specifică tipul operației care va fi efectuată:

```
% myisamchk optiuni nume_tabel ...
% isamchk optiuni nume_tabel ...
```

Un argument `nume_tabel` poate fi numele unui tabel sau numele fișierului index al unui tabel. Dacă specificați mai multe tabele, puteți folosi un model de nume de fișier pentru a selecta toate fișierele relevante dintr-un catalog:

```
% myisamchk optiuni *.MYI
% isamchk optiuni *.ISM
```

Nu veți distruge un tabel indicând verificarea acestuia de către un alt program decât cel adecvat, dar programul nu va face nimic altceva decât să emită un mesaj de avertisment. De exemplu, prima din următoarele instrucțiuni va verifica toate tabelele MyISAM din catalogul curent, în timp ce a doua nu va face decât să afișeze mesaje:

```
% myisamchk *.MYI        Corect
% isamchk *.MYI          Greșit; tip de fișier incorect
```

Nici unul dintre cele două utilitare nu face presupuneri cu privire la amplasarea tabelelor, deci trebuie fie să rulați programele în catalogul care conține fișierele tabel, fie să specificați calea de acces spre tabele. Aceasta vă permite să copiați fișierele tabel într-un alt catalog și să utilizați copiile.

Verificarea tabelelor

Atât `myisamchk` cât și `isamchk` furnizează metode de verificare a tabelelor care variază din punct de vedere al profunzimii cu care este examinat tabelul. De regulă, metoda normală este suficientă. Dacă verificarea normală nu raportează erori, dar dumneavoastră suspectați existența unor pagube (poate deoarece interogările par să nu funcționeze în mod adecvat), puteți dori să efectuați o verificare mai aprofundată. Pentru a executa o verificare normală a unui tabel cu oricare din cele două utilitare, invocați-le fără nici o opțiune:

```
% myisamchk nume_tabel
% isamchk nume_tabel
```

Pentru a se executa o verificare amănunțită, folosiți opțiunea `--extend-check`. Această opțiune poate fi foarte lentă, dar execută o verificare extrem de atentă; pentru fiecare înregistrare din fișierul de date al tabelului, este verificată cheia asociată pentru fiecare index din fișierul index, pentru a exista siguranța că indică într-adevăr spre înregistrarea adecvată. `myisamchk` mai are o opțiune intermediară, și anume `--medium-check`, care execută o verificare mai puțin aprofundată decât verificarea amănunțită, dar mai rapidă decât aceasta.

Dacă la o verificare cu opțiunea `--extend-check` nu sunt semnalate erori, puteți fi sigur că tabelul dumneavoastră nu are nici o problemă. Dacă tot mai aveți probleme cu tabelul, cauza trebuie să fie în altă parte. Reexaminați toate interogările care par să determine rezultate problematice, pentru a verifica dacă sunt scrise corect. Dacă sunteți de părere că problema poate veni din partea serverului MySQL, completați un raport privind hibele sau treceți la o versiune mai recentă.

Dacă `myisamchk` sau `isamchk` raportează existența unor erori în tabel, trebuie să le reparați folosind instrucțiunile din secțiunea următoare.

Repararea tabelelor

Repararea tabelelor este o treabă urâtă, mai ales datorită faptului că detaliile au tendința de a fi foarte specifice fiecărui incident în parte. Cu toate acestea, există îndrumări și proceduri de ordin general pe care le puteți respecta pentru a vă mări în mod semnificativ șansele de a putea repara tabelele.

În general, puteți începe cu metoda cea mai rapidă de reparare, pentru a vedea dacă astfel puteți remedia pagubele. Dacă descoperiți că această metodă nu este suficientă, puteți ajunge până la metode mai aprofundate (dar mai lente) până când paguba a fost remediată sau până când nu mai puteți folosi metode și mai aprofundate. (În practică, majoritatea problemelor pot fi remediate fără a se trece la moduri de reparare mai aprofundate și mai lente.) Dacă tabelul nu poate fi reparat, atunci restaurați-l din copiile dumneavoastră de siguranță. Instrucțiunile pentru refacerea tabelelor folosind fișierele copie de siguranță și jurnalele de actualizare se găsesc în capitolul 11.

Efectuarea reparației standard a unui tabel

Pentru a repara un tabel, parcurgeți următoarele etape:

1. Încercați să remediați tabelul folosind opțiunea `--recover`, dar folosiți și opțiunea `--quick` pentru a încerca să refaceți tabelul numai în funcție de conținutul fișierului index. Astfel, tabelul va fi reparat fără nici o interacțiune cu fișierul de date:

```
% myisamchk --recover --quick nume_tabel
% isamchk --recover --quick nume_tabel
```

2. Dacă problemele persistă, încercați comanda prezentată în etapa anterioară, dar omiteți opțiunea `--quick`, pentru a permite utilitarelor `myisamchk` și `isamchk` să modifice și fișierul de date:

```
% myisamchk --recover nume_tabel
% isamchk --recover nume_tabel
```

3. Dacă această metodă nu reușește, încercați modul de reparare `--safe-recover`. Acesta este mai lent decât modul de refacere obișnuit, dar este capabil de a remedia unele probleme pe care modul `--recover` nu le poate rezolva:

```
% myisamchk --safe-recover nume_tabel
% isamchk --safe-recover nume_tabel
```

Dacă utilitarele `myisamchk` sau `isamchk` se opresc în timpul oricăreia din aceste etape, cu un mesaj de eroare de forma `"Can't create new temp file: nume_fisier"` (nu poate crea un nou fișier temporar: *nume_fișier*), trebuie să repetați comanda și să adăugați opțiunea `--force`, pentru a forța eliminarea fișierului temporar. Probabil că acesta a fost „uitat" de la o încercare de reparare anterioară.

Copiați tabelele înainte de a le repara

O măsură generală de precauție pe care trebuie să o luați înainte de a efectua repararea tabelelor este de a efectua copii ale fișierelor tabel, în eventualitatea în care lucrurile scapă de sub control. Este puțin probabil să se întâmple așa ceva, dar *dacă* se întâmplă, puteți crea o nouă copie a tabelului din fișierele copiate și puteți încerca o altă metodă de refacere.

Ce este de făcut când metoda standard de reparare a tabelelor nu reușește

Dacă procedura de reparare standard nu reușește să repare tabelul, este posibil ca fișierul index să lipsească sau să fie imposibil de reparat. De asemenea, este posibil, deși puțin probabil, ca fișierul de descriere a tabelului să fie inexistent. În ambele cazuri va trebui să înlocuiți fișierele afectate, după care încercați din nou procedura de reparare standard.

Pentru a regenera fișierul index, folosiți următoarea procedură:

1. Deplasați-vă la catalogul de baze de date care conține tabelul deteriorat.
2. Deplasați fișierul de date al tabelului într-un loc sigur.
3. Invocați `mysql` și creați un nou tabel vid prin executarea următoarei instrucțiuni, care folosește fișierul de descriere a tabelului `nume_tabel.frm` pentru a regenera noul fișier de date, respectiv noul fișier index, pornind de la zero:

   ```
   mysql DELETE FROM nume_tabel;
   ```
4. Părăsiți programul `mysql` și deplasați fișierul de date original înapoi în catalogul de baze de date, înlocuind noul fișier de date vid pe care l-ați creat.
5. Încercați din nou să reparați tabelul prin metoda standard.

Pentru a reface fișierul de descriere a tabelului, restaurați-l pornind de la fișierele copie de siguranță, apoi încercați din nou procedura de reparare standard. Dacă, dintr-un motiv sau altul, nu dispuneți de o copie de siguranță, dar cunoașteți instrucțiunea `CREATE TABLE` care trebuie folosită pentru a crea tabelul, atunci mai aveți o șansă de a repara tabelul:

1. Deplasați-vă la catalogul de baze de date care conține tabelul deteriorat.
2. Deplasați fișierul de date al tabelului într-un loc sigur. Deplasați și fișierul index, dacă doriți să încercați să-l folosiți.
3. Invocați programul `mysql` și emiteți instrucțiunea `CREATE TABLE` care creează tabelul.
4. Părăsiți programul `mysql` și deplasați fișierul de date original înapoi în catalogul de baze de date, înlocuind noul fișier de date pe care l-ați creat. Dacă ați deplasat fișierul index în etapa 2, mutați fișierul index înapoi în catalogul de baze de date.
5. Încercați din nou să reparați tabelul prin metoda standard.

Evitarea interacțiunilor cu serverul MySQL

Când rulați un utilitar de verificare-reparare a tabelelor, nu este de dorit ca serverul MySQL și utilitarul să obțină simultan acces la un tabel. Evident că nu este bine deloc ca ambele programe să scrie în tabel, dar nu este mai bine nici dacă un program citește, iar celălalt scrie. Programul care citește poate fi derutat dacă tabelul este modificat în același timp de către celălalt program.

Puteți garanta inexistența oricărei interacțiuni între server și `myisamchk` sau `isamchk` dacă opriți serverul. Administratorii sunt însă reticenți (ceea ce este de înțeles) în ceea ce privește scoaterea completă din funcțiune a serverului, deoarece astfel devin indisponibile și bazele de date și tabelele fără defecțiuni. Procedurile descrise în această secțiune vă vor ajuta să evitați interacțiunea dintre server și `myisamchk` și `isamchk`.

Un server dispune de două categorii de blocări. Blocarea internă se folosește pentru a împiedica interferența dintre cererile clienților, de exemplu pentru a împiedica întreruperea interogării `SELECT` a unui client de către interogarea `UPDATE` a unui alt client. De asemenea, serverul folosește blocarea externă (blocare la nivelul sistemului de fișiere) pentru a preveni modificarea fișierelor tabel de către alte programe în timp ce fișierele respective sunt folosite de către server. În mod normal, serverul folosește blocarea externă pentru a coopera cu `myisamchk` și `isamchk` în timpul operațiilor de verificare a tabelelor. Totuși, blocarea externă este dezactivată pe unele sisteme, deoarece nu funcționează în mod fiabil. Procedura pe care trebuie să o alegeți pentru rularea programelor `myisamchk` și `isamchk` depinde de utilizarea sau nu a blocării externe de către server. Dacă serverul nu folosește blocarea externă, trebuie să folosiți un protocol de blocare internă.

Blocarea externă este dezactivată dacă serverul rulează cu opțiunea `--skip-locking`. Această opțiune este prestabilită pe unele sisteme, precum Linux. Puteți determina dacă serverul este sau nu capabil să folosească blocarea externă prin rularea comenzii `mysqladmin variables`. Verificați valoarea variabilei `skip-locking` și procedați după cum urmează:

- Dacă variabila `skip-locking` este dezactivată, blocarea externă este activată. Puteți folosi oricare din cele două utilitare pentru verificarea tabelelor. Serverul și utilitarul vor coopera în vederea accesului la tabele. Totuși, trebuie să evacuați zona cache pentru tabele cu ajutorul comenzii `mysqladmin flush-tables` înainte de a rula vreunul din utilitare. Pentru repararea tabelelor, trebuie să folosiți protocolul de blocare pentru repararea tabelelor.
- Dacă variabila `skip-locking` este activată, blocarea externă este dezactivată și serverul nu va ști dacă `myisamchk` sau `isamchk` verifică sau repară un tabel. Cel mai indicat este să opriți serverul. Dacă insistați să lăsați serverul în funcțiune, trebuie să vă asigurați că nici un client nu obține acces la tabel în timp ce dumneavoastră folosiți tabelul. Nu uitați să folosiți un protocol de blocare adecvat, pentru a indica serverului să lase tabelul în pace și să blocheze accesul clienților la tabel.

Protocoalele de blocare descrise aici folosesc mecanismul intern de blocare al serverului, pentru a-l împiedica pe acesta să obțină accesul la un tabel în timp ce dumneavoastră lucrați la tabelul respectiv cu `myisamchk` sau `isamchk`. Ideea generală este că invocați `mysql` și emiteți o instrucțiune `LOCK TABLE` pentru tabelul pe care doriți să-l verificați sau să-l reparați. Apoi, cu programul `mysql` în stare de inactivitate (adică rulează, dar nu face decât să mențină tabelul blocat), rulați `myisamchk` sau `isamchk`. După ce aceste utilitare și-au încheiat execuția, puteți reveni la sesiunea programului `mysql` și anulați blocarea, pentru a indica serverului că poate folosi din nou tabelul.

Protocoalele de blocare pentru verificare și reparare sunt oarecum diferite. Pentru verificare, trebuie să obțineți numai o blocare pentru citire. În acest caz, citiți tabelul, dar nu-l modificați, deci puteți permite și altor clienți să-l citească. Blocarea pentru citire este suficientă pentru a-i împiedica pe alți clienți să modifice tabelul. Pentru reparare, trebuie să vă procurați o blocare pentru scriere, cu scopul de a-i împiedica pe alți clienți să modifice tabelul în timp ce dumneavoastră lucrați la tabel.

Protocoalele de blocare folosesc instrucțiunile `LOCK TABLE` și `UNLOCK TABLE` pentru a dobândi, respectiv pentru anula blocările. De asemenea, protocoalele folosesc `FLUSH TABLES` pentru a evacua pe disc toate modificările suspendate și pentru a redeschide tabelul după ce acesta a fost modificat de un utilitar de reparare a tabelului. *Trebuie* să efectuați toate instrucțiunile `LOCK`, `FLUSH` și `UNLOCK` în cadrul unei singure sesiuni `mysql`. Dacă blocați un tabel și apoi părăsiți programul `mysql`, blocarea va fi anulată și rularea utilitarelor `myisamchk` și `isamchk` nu se va desfășura în condiții de siguranță!

Procedurile de blocare se pot desfășura cel mai ușor dacă păstrați deschise două ferestre: una pentru rularea programului `mysql` și alta pentru rularea programelor `myisamchk` sau `isamchk`, ceea ce vă permite să comutați cu ușurință de la un program la altul. Dacă nu lucrați într-un mediu cu ferestre, trebuie să suspendați și să reluați execuția programului `mysql` folosind funcționalitățile de control al sarcinilor caracteristice interpretorului în timp ce rulați `myisamchk` sau `isamchk`. Instrucțiunile următoare prezintă comenzile pentru `myisamchk` și `isamchk`; folosiți-le pe cele care sunt adecvate pentru tabelul sau tabelele la care lucrați.

Blocarea unui tabel în vederea operațiilor de verificare

Această procedură se folosește numai pentru verificarea unui tabel, nu pentru repararea acestuia. În fereastra 1, invocați `mysql` și emiteți următoarele instrucțiuni:

```
% mysql nume_baza_de_date
mysql> LOCK TABLE nume_tabel READ;
mysql> FLUSH TABLES;
```

Blocarea împiedică pe alți clienți să scrie în tabel și să modifice tabelul în timp ce dumneavoastră îl verificați. Instrucțiunea `FLUSH` determină serverul să închidă fișierele tabel, ceea ce determină evacuarea tuturor modificărilor care mai pot exista în cache.

Cu programul `mysql` inactiv, comutați în fereastra 2 și verificați tabelul:

```
% myisamchk nume_tabel
% isamchk nume_tabel
```

Când `myisamchk` sau `isamchk` și-a încheiat execuția, reveniți la sesiunea `mysql` în fereastra 1 și anulați blocarea tabelului:

```
mysql> UNLOCK TABLE;
```

Dacă `myisamchk` sau `isamchk` raportează identificarea unei probleme cu tabelul, va trebui să efectuați repararea tabelului.

Blocarea unui tabel în vederea operațiilor de reparare

Procedura de blocare pentru repararea unui tabel este similară cu procedura pentru verificarea acestuia, cu două diferențe. Mai întâi, trebuie să obțineți o blocare pentru scriere, nu una pentru citire. Veți modifica tabelul, deci nu puteți permite sub nici o formă clienților accesul la acesta. În al doilea rând, trebuie să emiteți o instrucțiune `FLUSH TABLES` după efectuarea reparațiilor, deoarece `myisamchk` și `isamchk` construiesc un nou fișier index, dar serverul nu va remarca acest lucru decât dacă evacuați din nou zona cache pentru tabele. Pentru a executa această procedură, invocați `mysql` în fereastra 1 și emiteți următoarele instrucțiuni:

```
% mysql nume_baza_de_date
mysql> LOCK TABLE nume_tabel WRITE;
mysql> FLUSH TABLES;
```

Cu programul `mysql` inactiv, faceți copii ale fișierelor tabel, dacă nu ați procedat deja astfel, apoi rulați `myisamchk` sau `isamchk`:

```
% cp nume_tabel.* /un/alt/catalog
% myisamchk --recover nume_tabel
% isamchk --recover nume_tabel
```


Opțiunea `--recover` nu are decât un caracter ilustrativ. Opțiunile particulare pe care le alegeți vor depinde de tipul de reparație pe care îl executați. După ce programele `myisamchk` sau `isamchk` și-au încheiat execuția, reveniți la sesiunea `mysql` din fereastra 1, evacuați din nou zona de memorie cache pentru tabele și anulați blocarea tabelului:

```
mysql> FLUSH TABLES;
mysql> UNLOCK TABLE;
```

Mărirea vitezei de lucru a programelor myisamchk și isamchk

Rularea programelor `myisamchk` și `isamchk` poate necesita un timp îndelungat, mai ales dacă lucrați cu un tabel mare sau folosiți una dintre metodele mai amănunțite de verificare sau de reparare. Puteți mări viteza de lucru a acestor programe, indicându-le să folosească o cantitate mai mare de memorie atunci când rulează.

Reducerea timpului de inactivitate a serverului

O altă modalitate de a împiedica serverul să obțină accesul la tabelul unde lucrați este de a utiliza copii ale fișierelor tabel, amplasate în afara catalogului de date. Totuși, problema cu interacțiunea nu este eliminată, deoarece trebuie în continuare să împiedicați serverul să obțină accesul la tabel (și eventual chiar să-l modifice!) în timp ce dumneavoastră efectuați copiile. Totuși, aceasta este o metodă de a reduce timpul de inactivitate a serverului și poate fi interesantă pentru dumneavoastră, dacă sunteți reticent în a scoate complet serverul din funcțiune. Opriți serverul pentru o scurtă perioadă de timp, atunci când copiați fișierele tabel de care aveți nevoie într-un alt catalog, după care reporniți serverul.

Planuri de viitor pentru myisamchk

Se intenționează ca funcțiile de verificare și reparare a tabelelor de care dispune utilitarul `myisamchk` să fie fuzionate în server într-un anumit moment din dezvoltarea seriei de versiuni MySQL 3.23. Când acest lucru se va întâmpla, verificarea și repararea tabelelor vor deveni mult mai simple, deoarece nu vor mai apărea problemele legate de interacțiunea serverului cu programul `myisamchk`.

De asemenea, veți putea cere serverului să verifice tabelele la pornire, deci nu va fi necesară stabilirea unor comenzi speciale ce ar urma să fie executate la încărcarea sistemului de operare, înainte de pornirea serverului. Totuși, această situație nu este valabilă pentru tabelele ISAM, deci, atunci când serverul va fi dotat cu funcționalitățile de verificare și reparare, v-aș recomanda să aveți în vedere conversia tabelelor ISAM în tabele MyISAM. Examinați manualul de referință MySQL al noilor versiuni, pentru a vedea evoluțiile care au loc în acest domeniu. Puteți converti un tabel folosind o instrucțiune `ALTER TABLE`:

```
ALTER TABLE nume_tabel TYPE = MYISAM
```

Ambele utilitare dispun de numeroși parametri de operare care pot fi configurați. Cele mai importante din aceste variabile controlează dimensiunile bufferelor folosite de către programe:

Variabilă	Semnificație
`key_buffer_size`	Dimensiunea bufferului folosit pentru stocarea blocurilor de index
`read_buffer_size`	Dimensiunea bufferului folosit pentru operațiile de citire
`sort_buffer_size`	Dimensiunea bufferului folosit pentru sortare
`write_buffer_size`	Dimensiunea bufferului folosit pentru operațiile de scriere

Pentru a afla valorile folosite în mod prestabilit de fiecare program pentru aceste variabile, rulaţi programul cu opţiunea `--help`. Pentru a specifica o altă valoare, folosiţi `--set-variable` *`variabila=valoare`* sau `-O` *`variabila=valoare`* în linia de comandă. Puteţi prescurta numele variabilelor sub forma `key`, `read`, `sort` şi `write`. De exemplu, pentru a indica programului `myisamchk` să folosească un buffer de sortare de 16MB şi buffere de citire, respectiv scriere, de 1MB, invocaţi programul astfel:

```
% myisamchk -O sort=16M -O read=1M -O write=1M ...
```

Variabila `sort_buffer_size` este folosită numai cu opţiunea `--recover` (nu cu opţiunea `--safe-recover`), caz în care variabila `key_buffer` nu este folosită.

Planificarea întreţinerii preventive

Trebuie să aveţi în vedere alcătuirea unei planificări a operaţiilor de întreţinere preventivă, pentru a contribui la detectarea automată a problemelor, astfel încât să puteţi lua măsuri de remediere a acestora:

1. Executaţi sistematic copii de siguranţă ale bazelor de date şi activaţi caracteristica de consemnare în jurnalele de actualizare. Nu-i aşa c-am mai spus-o?
2. Stabiliţi sistematic o verificare periodică a tabelelor. Prin verificarea tabelelor dumneavoastră, veţi diminua posibilitatea de a fi obligat să vă folosiţi copiile de siguranţă. Această operaţie se poate executa cel mai simplu sub forma unei sarcini a utilitarului `cron`, invocată în mod caracteristic din fişierul `crontab` al contului folosit pentru rularea serverului. De exemplu, dacă rulaţi serverul ca utilizator `mysqladm`, atunci puteţi stabili o verificare periodică din fişierul `crontab` pentru `mysqladm`. Dacă nu ştiţi să folosiţi `cron`, examinaţi paginile aferente din manualul UNIX folosind aceste comenzi:

   ```
   % man cron
   % man crontab
   ```

3. Verificaţi tabelele din bazele de date în timpul procedurii de încărcare a sistemului de operare, înainte de pornirea serverului. S-ar putea ca reîncărcarea sistemului de operare să aibă loc datorită unei căderi anterioare a sistemului. În acest caz, este posibil ca tabelele din baza de date să fie deteriorate şi trebuie verificate.

Pentru a executa o verificare automată a tabelelor, puteţi scrie un script care comută în catalogul de date al serverului şi rulează `myisamchk` şi `isamchk` pe toate tabelele din baza dumneavoastră de date. În scripturile prezentate în expunerea următoare, vom folosi ambele programe. Dacă dispuneţi numai de tabele MyISAM sau numai de tabele ISAM, aveţi nevoie numai de unul din aceste programe şi îl puteţi omite pe celălalt din scripturi.

Atât `myisamchk`, cât şi `isamchk` generează date de ieşire în modul de verificare a tabelelor, pentru a vă informa care sunt tabelele pe care le verifică, chiar dacă nu există nici un fel de probleme. Pentru intrările `crontab`, în mod normal este de dorit ca datele de ieşire să fie suprimate, cu excepţia situaţiilor când apar erori în tabele. (În mod caracteristic, sarcinile utilitarului `cron` generează un mesaj de poştă dacă vreo sarcină produce date de ieşire, şi nu există prea multe motive să primiţi mesaje pentru sarcini de verificare a tabelelor care nu găsesc nici o problemă.) Dacă invocaţi orice utilitar cu opţiunea `--silent`, va genera date de ieşire numai la apariţia problemelor. De asemenea, `myisamchk` acceptă o opţiune `--fast`, care îi permite să omită orice tabel care nu a fost modificat de la ultima verificare.

Un script simplu, care verifică toate tabelele din catalogul de date al serverului, se prezintă astfel (`DATADIR` trebuie înlocuit cu o valoare adecvată pentru sistemul dumneavoastră):

```
#! /bin/sh
cd DATADIR
myisamchk --silent --fast t */*.MYI
isamchk --silent */*.ISM
```

O potenţială problemă a acestui script este aceea că, dacă dispuneţi de numeroase tabele modelele cu caractere de înlocuire `*/*.MYI` sau `*/*.ISM` pot agasa interpretorul cu eror de genul „prea multe argumente". Un script alternativ poate fi scris astfel (din nou înlocuiţi `DATADIR` cu o valoare adecvată pentru sistemul dumneavoastră):

```
#! /bin/sh
datadir=DATADIR
find $datadir -name "*.MYI" -print | xargs myisamchk --silent --fast
find $datadir -name "*.ISM" -print | xargs isamchk --silent
```

Indiferent de forma de script pentru care optaţi, voi presupune că îl veţi denum `check_mysql_tables`. Nu uitaţi să-i modificaţi modul, astfel încât scriptul să fie executabi Apoi, trebuie să invocaţi scriptul manual, pentru a verifica dacă funcţionează corect:

```
% chmod +x check_mysql_tables
% check_mysql_tables
```

În mod ideal, nu vor exista date de ieşire. Dacă sistemul dumneavoastră nu acceptă blocare externă, este posibil ca serverul să modifice un tabel în timp ce îl verificaţi. În acest caz, scrip tul poate raporta probleme pentru tabele unde aceste probleme nu există. Situaţia nu este pre fericită, dar este de preferat cazului invers, când scriptul nu raportează probleme, dar aceste există. Dacă sistemul dumneavoastră acceptă blocarea externă, această problemă nu va apăre

Secţiunile următoare prezintă modul de configurare a scriptului astfel încât acesta să f executat automat de `cron` şi la pornirea sistemului. În exemplele din aceste secţiuni, v presupune că scriptul este instalat în catalogul `/usr/users/mysqladm/bin`. Va trebui modificaţi calea conform configuraţiei sistemului dumneavoastră.

Dacă rulaţi mai multe servere în sistemul dumneavoastră, va trebui să modificaţi proc dura astfel încât să verificaţi tabelele din catalogul de date al fiecărui server. Puteţi fa aceasta executând copii diferite ale scriptului `check_mysql_tables` sau prin modificar scriptului astfel încât să accepte un argument de linie de comandă care să specifice cat logul de date pe care doriţi să-l verificaţi.

Verificarea periodică a tabelelor cu ajutorul utilitarului cron

Să presupunem că doriţi să invocaţi scriptul `check_mysql_tables` din fişierul `cront` pentru utilizatorul `mysqladm`. Deschideţi sesiunea de lucru sub contul respectiv, apoi ed taţi fişierul dumneavoastră `crontab` folosind această comandă:

```
% crontab -e
```

Această comandă vă va duce într-un editor de texte, în care va insera o copie a fişierul `crontab` curent (care poate fi vid, dacă nu aţi mai executat această operaţie în prealabi Adăugaţi la fişier o linie care se prezintă astfel:

```
0 3 * * 0 /usr/users/mysqladm/bin/check_mysql_tables
```

Această intrare indică utilitarului cron să ruleze comanda în fiecare duminică, la ora 3 dimineața. Puteți modifica ora sau planificarea după dorință. Căutați în pagina de manual crontab pentru a determina formatul intrărilor.

Verificarea tabelelor la pornirea sistemului

Dacă folosiți un sistem de tip BSD și ați adăugat comanda de pornire a serverului la fișierul /etc/rc.local sau la un echivalent al acestuia, puteți pur și simplu invoca check_mysql_tables din fișierul respectiv, înainte de a porni serverul.

Dacă folosiți metoda de pornire caracteristică sistemelor de tip System V, care invocă mysql.server din unul din cataloagele /etc/rc.d, procedura este ceva mai complexă. Scripturile de pornire din aceste cataloage trebuie să înțeleagă argumentele start și stop, astfel încât să poată întreprinde acțiuni adecvate la pornirea, respectiv oprirea sistemului. Pentru a executa verificarea tabelelor, putem scrie un script care invocă check_mysql_tables atunci când argumentul este start și care nu face nimic atunci când argumentul este stop. Să denumim scriptul mysql.check; aspectul său este următorul:

```
#! /bin/sh

# Sa vedem cum am fost apelati.
case "$1" in
  start)
     echo -n "Verifica tabelele MySQL: "
     if [ -x /usr/users/mysqladm/bin/check_mysql_tables ]; then
          /usr/users/mysqladm/bin/check_mysql_tables
     fi
     ;;
  stop)
     # nu face nimic
     ;;
  *)
     echo "Usage: $0 {start|stop}"
     exit 1
esac

exit 0
```

Acum, puteți instala mysql.check folosind o procedură similară celei descrise în capitolul 11 pentru instalarea scriptului mysql.server. Nu uitați să atribuiți scriptului mysql.check un număr de prefix inferior în catalogul la nivel de rulare, astfel încât să ruleze înainte de mysql.server. De exemplu, dacă vă legați la mysql.server ca S99mysql.server din catalogul la nivel de rulare, vă puteți lega la mysql.check ca S98mysql.check.

PARTEA A IV-A

Anexe

ANEXA A

Obținerea și instalarea programelor

Această anexă descrie modul de obținere și instalare a programelor de care veți avea nevoie pentru a rula MySQL și programele corelate create de terțe părți descrise în această carte, precum suportul pentru Perl DBI și CGI.pm, PHP și Apache. De asemenea, este prezentat modul de obținere a fișierelor asociate cu baza de date demonstrativă samp_db folosită pe parcursul acestei cărți.

Scopul acestui capitol este de a aduna la un loc instrucțiunile de instalare pentru toate pachetele, nu de a înlocui instrucțiunile incluse în fiecare pachet. De fapt, vă recomand să citiți aceste instrucțiuni. Anexa de față furnizează instrucțiuni de ordin general care ar trebui să fie suficiente în majoritatea situațiilor, dar fiecare distribuție software conține și instrucțiuni care vă ajută să remediați problemele în cazul eșecului unei proceduri de instalare standard. De exemplu, manualul MySQL conține un capitol de mari dimensiuni dedicat procedurilor de instalare și care include soluții pentru numeroase probleme specifice sistemului.

Selectarea programelor de instalat

Trebuie să instalați MySQL dacă nu ați procedat deja astfel, dar este necesar să instalați numai acele instrumente create de terțe părți pe care doriți să le utilizați:

- Dacă doriți să scrieți scripturi Perl care obțin acces la bazele de date MySQL, trebuie să instalați suportul MySQL pentru programarea clienților și modulele DBI. Dacă intenționați să scrieți scripturi DBI care furnizează acces prin Web la bazele dumneavoastră de date, probabil că veți dori să instalați modulul CGI.pm și veți avea nevoie de un server Web. În această carte este folosit serverul Apache, dar se pot folosi și alte servere.
- Dacă doriți să scrieți scripturi PHP, trebuie să instalați suport de programare a clienților și PHP. În mod normal, PHP este folosit pentru activitatea de scripting în Web, ceea ce înseamnă că mai aveți nevoie și de un server Web. Serverul Apache este folosit în această carte deoarece Apache are suport modular pentru PHP. Aceasta înseamnă că puteți instala PHP ca parte încorporată a binarului dumneavoastră Apache. Dacă nu puteți modifica exemplarul dumneavoastră al serverului Apache (de exemplu dacă vă aflați pe o gazdă ISP și ISP controlează Apache) sau dacă rulați un alt server Web, puteți construi PHP ca program autonom și rulați scripturi PHP sub formă de scripturi CGI standard. Instalarea PHP ca modul Apache asigură performanțe mai bune decât rularea acestuia ca program autonom.

Fișierele binare precompilate sunt disponibile pentru multe dintre pachetele de instalare. Pentru Linux sunt disponibile mai multe fișiere RPM. Dacă preferați să compilați programe de la sursă sau dacă nu este disponibilă o distribuție binară pentru platforma dumneavoastră, veți avea nevoie de un compilator C (C++ pentru MySQL). Dezvoltatorii MySQL recomandă utilizarea compilatorului gcc 2.95.1 sau a unei versiuni mai recente a acestuia pentru a construi MySQL. De asemenea, se pot folosi și versiunile compilatorului egcs începând de la numărul 1.03.

Alte instrumente de care veți avea nevoie includ gunzip și tar (dacă instalați din fișiere tar comprimate), respectiv rpm (dacă instalați din fișiere RPM).

Obținerea programelor

Principalele puncte de distribuție pentru fiecare dintre pachetele de care veți avea nevoie sunt prezentate în tabelul următor. Unele pachete pot fi obținute din mai multe locații:

Pachet	Locație	
MySQL	http://www.mysql.com/	ftp://ftp.mysql.com/pub/mysql/
DBI	http://www.symbolstone.org/technology/perl/DBI/	
	http://www.mysql.com/Contrib/	
PHP	http://www.php.net/	
	ftp://ftp.php.net/	
Apache	http://www.apache.org/	ftp://ftp.apache.org/
CGI.pm	http:/stein.cshl.org/WWW/software/CGI/	

Dacă folosiți FTP pentru transferul fișierelor, nu uitați să le transferați în mod binar, pentru a evita deteriorarea fișierelor.

Există multe situri oglindă pentru MySQL. www.mysql.com se află în Scandinavia, deci, dacă alegeți un sit oglindă care este mai apropiat de dumneavoastră, probabil că veți obține timpi de descărcare mai buni. Siturile oglindă sunt menționate la următoarea adresă:

http://www.mysql.com/mirrors.html

Pentru programele Perl, puteți folosi întotdeauna http://cpan.perl.org/ pentru a localiza siturile oglindă care conțin modulele de care aveți nevoie.

Selectarea versiunii care va fi instalată

Versiunea unui pachet pe care îl instalați depinde de necesitățile dumneavoastră:

- Dacă aveți nevoie de un maximum de stabilitate, probabil că trebuie să fiți conservator și să folosiți cea mai recentă versiune stabilă (nu beta) a unui pachet. Astfel, beneficiați de cele mai recente caracteristici și de cel mai mare număr de remedii ale hibelor, fără a vă expune programelor experimentale din versiunea beta.
- Dacă doriți să fiți mereu la modă sau dacă sunteți programator, poate doriți să folosiți cea mai recentă versiune beta. Desigur, dacă aveți nevoie în mod absolut de o caracteristică existentă numai într-o versiune beta, nu aveți altă opțiune decât să folosiți versiunea respectivă.

- Pentru MySQL, distribuțiile RPM și binare încorporate sunt deseori construite folosind indicatori de optimizare, care sunt mai buni decât ceea ce poate determina, de unul singur, scriptul de configurare din distribuția sursă. De asemenea, dezvoltatorii MySQL construiesc distribuții folosind unele compilatoare comerciale de optimizare, pentru a mări și mai mult viteza sistemului MySQL. În consecință, programele din aceste categorii de distribuții pot rula mai rapid decât cele pe care le compilați dumneavoastră. Dacă doriți să obțineți un maximum de performanță, puteți dori să încercați o distribuție încorporată.

Siturile Web pentru fiecare pachet indică versiunile care constituie cele mai recente variante stabile, precum și cele care reprezintă variante beta. De asemenea, aceste situri conțin liste cu modificările caracteristicilor pentru fiecare versiune în parte, pentru a vă ajuta să decideți care este varianta cea mai potrivită pentru dumneavoastră.

Dacă lucrați cu distribuții RPM sau binare, despachetarea unei distribuții este echivalentă cu instalarea acesteia, deoarece fișierele sunt despachetate în cataloagele unde doriți să se găsească acestea. Dacă o distribuție instalează fișiere în cataloage protejate, atunci trebuie să lucrați sub contul `root` pentru a despacheta distribuția.

Pentru distribuțiile sursă, le puteți despacheta într-o regiune pe care o folosiți pentru compilare și apoi instalați programul în locațiile de instalare. Pentru a efectua etapa de instalare este necesar să lucrați sub contul `root`, dar această calitate nu este necesară pentru nici una dintre etapele de configurare sau compilare.

Înscrierea în listele de corespondență pentru a obține ajutor

Când instalați un pachet, se recomandă să vă înscrieți pe lista de discuții generală pentru pachetul respectiv, astfel încât să puteți pune întrebări și primi răspunsuri utile. Dacă instalați variante beta, categoric trebuie să vă pregătiți pentru a vă înscrie în lista de corespondență asociată programului și pentru a citi mesajele din această listă, pentru a fi la curent cu rapoartele privind hibele și cu remediile apărute. Dacă nu vă înscrieți într-o listă de discuții generală, trebuie să vă înscrieți cel puțin în lista de anunțuri, pentru a primi înștiințări privind noile variante. În introducerea acestei cărți se găsesc instrucțiuni pentru înscrierea pe listele de corespondență și pentru utilizarea acestora. Siturile Web pentru fiecare pachet furnizează de asemenea informații privind înscrierea.

Dacă instalați de la o distribuție sursă, numeroase pachete sunt configurate cu ajutorul utilitarului `configure`, ceea ce facilitează configurarea și construirea programelor pe o varietate de sisteme. În cazul în care construirea unui program eșuează, puteți fi obligat să rulați din nou utilitarul `configure` cu alte opțiuni decât cele pe care le-ați specificat inițial. Dacă este necesar, trebuie să împiedicați utilitarul `configure` să folosească informații salvate de la rularea anterioară a acestui program. Puteți șterge astfel informațiile stocate:

```
% make distclean
```

Alternativ, mai puteți proceda și astfel:

```
% rm config.cache
% make clean
```

Indiferent de caz, trebuie să rulați din nou `configure` după aceea.

Instalarea sistemului MySQL pe UNIX

Există distribuții disponibile pentru diferite versiuni de MySQL. Variantele stabile curente au numere de versiune din seria 3.22. Variantele de dezvoltare curente se află în seria 3.23. În general, trebuie să folosiți versiunea cu numărul cel mai mare din seria pe care doriți să o folosiți disponibilă în formatul de distribuție pe care doriți să-l utilizați.

Distribuțiile MySQL se folosesc în format binar, RPM și format sursă. Distribuțiile binară și RPM sunt mai ușor de instalat, dar trebuie să acceptați parametrii prestabiliți pentru structura și configurarea instalării, care sunt încorporați în distribuție. Distribuțiile sursă sunt mai dificil de instalat, deoarece trebuie să compilați programul, dar dobândiți un control mai mare asupra parametrilor de configurare. De exemplu, puteți compila distribuția destinată numai suportului pentru clienți fără a construi serverul, iar dumneavoastră puteți modifica locația unde doriți să instalați programul.

Distribuțiile conțin una sau mai multe dintre următoarele componente:

- Serverul `mysqld`
- Programele client (`mysql`, `mysqladmin` și așa mai departe) și suport pentru programarea clienților (fișiere bibliotecă și antet)
- Documentație
- Baza de date pentru evaluarea performanțelor
- Suport pentru limbaj

Distribuțiile sursă și binară conțin toate aceste elemente. Fiecare fișier RPM conține numai câteva dintre ele, deci puteți fi obligat să instalați mai multe fișiere RPM pentru a obține tot ceea ce aveți nevoie.

Dacă intenționați să vă conectați la un server care rulează deja pe un alt calculator, nu trebuie să instalați un server. Dar trebuie întotdeauna să instalați programe client:

- Dacă nu rulați un server, veți avea nevoie de clienți pentru a vă putea conecta la un server situat pe un alt calculator.
- Dacă rulați un server, doriți să vă puteți conecta la acesta de la gazda serverului, nu să fiți forțat să deschideți sesiunea de lucru la un alt calculator care dispune de programul client, numai pentru a vă testa serverul.

O trecere în revistă a instalării sistemului MySQL

Instalarea sistemului MySQL pe UNIX implică următoarele etape:

1. Creați un cont UNIX pentru utilizatorul și grupul sub al cărui cont va rula utilizatorul (dacă instalați serverul).
2. Obțineți și despachetați toate distribuțiile pe care doriți să le instalați. Dacă folosiți o distribuție sursă, compilați-o și instalați-o.
3. Rulați scriptul `mysql_install_db` pentru a inițializa catalogul de date și tabelele de acordare (numai când sistemul MySQL este instalat pentru prima dată pe un calculator).
4. Porniți serverul.

5. Citiți capitolul 11, „Administrarea generală a sistemului MySQL", pentru a vă familiariza cu procedurile administrative generale. În particular, trebuie să citiți secțiunile cu privire la pornirea și oprirea serverului, precum și la rularea serverului în calitate de utilizator fără privilegii.

Crearea unui cont pentru utilizatorul MySQL

Puteți omite această secțiune dacă rulați numai programe client MySQL și nu intenționați să rulați un server MySQL.

Serverul MySQL poate fi rulat sub orice cont de utilizator UNIX pe sistemul dumneavoastră, dar, din motive administrative și de securitate, cel mai bine este să nu rulați serverul sub contul `root`. Vă recomand să creați un cont separat pentru activitatea de administrare a sistemului MySQL și să rulați serverul sub contul respectiv. În acest mod, puteți deschide sesiunea de lucru sub contul utilizatorului respectiv și aveți privilegii complete în catalogul de date pentru executarea operațiilor de întreținere și depanare. Procedurile pentru crearea conturilor de utilizator variază de la un sistem la altul. Consultați-vă documentația locală, pentru a obține detalii mai concrete.

Această carte folosește `mysqladm` și `mysqlgrp` ca nume de utilizator și nume de grup UNIX ale acestui cont. Dacă intenționați să instalați MySQL numai pentru uz personal, îl puteți rula sub contul propriu, caz în care veți putea folosi propriile dumneavoastră nume de utilizator și de grup oriunde vedeți mențiunile `mysqladm` și `mysqlgrp` în această carte. Dacă instalați din fișiere RPM, un utilizator denumit `mysql` este creat automat ca parte a procedurii de instalare RPM. În acest caz, veți înlocui `mysqladm` cu `mysql`.

Avantajele utilizării unui cont separat, fără privilegii, în locul contului `root` pentru exploatarea sistemului MySQL sunt următoarele:

- Dacă nu rulați MySQL ca `root`, nimeni nu poate exploata serverul ca breșă de securitate pentru a dobândi accesul sub contul `root`.
- Executarea operațiunilor administrative din MySQL din contul unui utilizator fără privilegii este mai sigură decât din contul `root`.
- Serverul va crea fișiere aflate în posesia utilizatorului `mysqladm`, nu în cea a utilizatorului `root`. Cu cât utilizatorul `root` deține mai puține fișiere în sistemul dumneavoastră, cu atât mai bine.
- Din punct de vedere conceptual, este mai indicat ca activitatea legată de MySQL să se desfășoare într-un cont separat; astfel, aspectele legate de MySQL din sistemul dumneavoastră devin mai ușor de sesizat. De exemplu, în catalogul unde sunt păstrate fișierele `cron`, veți avea un fișier separat pentru utilizatorul MySQL `mysqladm`. În caz contrar, sarcinile `cron` aferente sistemului MySQL se vor găsi în fișierul utilizatorului `root`, alături de toate operațiile executate periodic în contul `root`.

Obținerea și instalarea distribuției MySQL

În instrucțiunile următoare, voi folosi `versiune` ca simbol al numărului versiunii distribuției MySQL pe care o utilizați, respectiv `platforma` ca nume al platformei pe care o instalați. Acestea sunt folosite în numele de fișiere din distribuție, astfel încât dis-

tribuțiile să poată fi identificate cu ușurință și diferențiate una de alta. Un număr de versiune este ceva de genul `3.22.26` sau `3.23.4-alpha`. Un nume de platformă este ceva de genul `sgi-irix6.3-mips` sau `dec-osf4.0b-alpha`.

Instalarea unei distribuții binare

Fișierele unei distribuții binare au nume precum `mysql-versiune-platforma.tar.gz`. Procurați-vă fișierul de distribuție pentru versiunea și platforma dorite și inserați-l în catalogul unde doriți să instalați MySQL, de exemplu `/usr/local`.

Despachetați distribuția folosind una dintre următoarele comenzi (folosiți a doua comandă, dacă versiunea dumneavoastră a programului `tar` nu înțelege opțiunea `z` pentru decomprimarea distribuției):

```
% tar zxf mysql-versiune-platforma.tar.gz
% gunzip < mysql-versiune-platforma.tar.gz | tar xf -
```

Prin despachetarea distribuției se creează un catalog, denumit `mysql-versiune-platforma`, care include conținutul distribuției. Pentru a facilita referirea la acest catalog, creați o legătură simbolică:

```
% ln -s mysql-versiune-platforma mysql
```

Acum, puteți face referire la catalogul de instalare sub numele `/usr/local/mysql`, dacă ați instalat MySQL în catalogul `/usr/local`.

Dacă veți folosi numai suportul pentru clienți furnizat de distribuție și nu rulați un server, ați terminat de instalat sistemul MySQL. Dacă instalați MySQL pentru prima dată, totuși, consultați secțiunea „Inițializarea catalogului de date și a tabelelor de acordare". Dacă actualizați o instalare existentă, consultați secțiunea „Pornirea serverului".

Instalarea unei distribuții RPM

Fișierele RPM sunt disponibile pentru instalarea sistemului MySQL pe sisteme Linux. Numele de fișier ale acestora sunt următoarele:

- `MySQL-versiune-platforma.rpm`

  Programul server.
- `MySQL-client.versiune-platforma.rpm`

  Programele client.
- `MySQL-devel.versiune-platforma.rpm`

  Suport pentru dezvoltare (biblioteci client și fișiere antet) pentru scrierea programelor client. Veți avea nevoie de acestea dacă doriți să folosiți sau să scrieți scripturi Perl DB pentru accesul la bazele de date MySQL.
- `MySQL-bench.versiune-platforma.rpm`

  Programe de testare și evaluare a performanțelor. Necesită Perl și modulul `Msql Mysql-modules`. Vă puteți procura un RPM pentru modulul `Msql-Mysql-modules` de la adresa `www.mysql.com`.
- `MySQL-versiune.src.rpm`

  Sursa pentru programul server, programele client și programele de testare și evaluare a performanțelor.

Nu trebuie să vă aflați într-un anumit catalog atunci când instalați MySQL dintr-o distribuție RPM, deoarece fișierele RPM conțin informații care arată unde trebuie instalate fișierele pe care le conțin. Pentru orice fișier RPM `fisier_rpm` puteți afla unde va fi instalat conținutul acestuia cu ajutorul următoarei comenzi:

```
% rpm -qpl fisier_rpm
```

Pentru a instala un fișier RPM, folosiți această comandă:

```
% rpm -i fisier_rpm
```

Diferitele componente ale sistemului MySQL sunt împărțite în diferite fișiere RPM, deci poate fi necesar să instalați mai mult de un fișier RPM. Pentru a instala suportul pentru clienți, folosiți această comandă:

```
% rpm -i MySQL-client-versiune-platforma.rpm
```

Pentru a instala suportul pentru server, folosiți următoarea comandă:

```
% rpm -i MySQL-versiune-platforma.rpm
```

Dacă intenționați să scrieți propriile dumneavoastră programe folosind suportul pentru programarea clienților, nu uitați să instalați fișierul RPM de dezvoltare:

```
% rpm -i MySQL-devel-versiune-platforma.rpm
```

Dacă intenționați să folosiți numai suportul pentru clienți furnizat de distribuție și nu rulați un server, ați terminat de instalat sistemul MySQL. Dacă instalați MySQL pentru prima dată, totuși, consultați secțiunea „Inițializarea catalogului de date și a tabelelor de acordare". Dacă actualizați o instalare existentă, consultați secțiunea „Pornirea serverului".

Dacă doriți să instalați de la fișierul RPM sursă, următoarea comandă este suficientă:

```
% rpm --recompile MySQL-versiune.src.rpm
```

Instalarea unei distribuții sursă

Distribuțiile sursă au nume precum `mysql-versiune.tar.gz`, unde `versiune` este numărul versiunii MySQL. Selectați catalogul unde doriți să despachetați distribuția și deplasați-vă la acesta. Procurați-vă fișierul de distribuție și despachetați-l folosind una dintre următoarele comenzi (folosiți a doua comandă în cazul în care versiunea dumneavoastră a programului `tar` nu înțelege opțiunea z pentru decomprimarea distribuției):

```
% tar zxf mysql-versiune-platforma.tar.gz
% gunzip < mysql-versiune-platforma.tar.gz | tar xf -
```

Prin despachetarea distribuției, se creează un catalog `mysql-versiune`, care include conținutul distribuției. Deplasați-vă în acest catalog:

```
% cd mysql-versiune
```

Acum, trebuie să configurați și să compilați distribuția înainte de a o putea instala. Dacă aceste etape nu dau nici un rezultat, examinați capitolul „Instalarea sistemului MySQL" din manualul de referință MySQL, cu precădere orice note specifice de sistem pe care le poate conține manualul cu privire la tipul calculatorului dumneavoastră.

Folosiți comanda `configure` pentru a configura distribuția:

```
% ./configure
```

Puteți specifica opțiuni pentru `configure`, dacă doriți. Pentru a obține o listă cu opțiunile disponibile, rulați următoarea comandă:


```
% ./configure --help
```

Lista următoare prezintă unele opțiuni de configurare utile pentru mulți:

- `--without-server`

  Configurare numai pentru construirea suportului pentru clienți (programe client sau biblioteci client). Puteți folosi această opțiune dacă intenționați să obțineți acces la un server care rulează deja pe un alt calculator.

- `--prefix=nume_cale`

  În mod prestabilit, catalogul rădăcină folosit la instalare este `/usr/local`. Catalogul de date, clienții, serverul, bibliotecile client și fișierele antet sunt instalate respectiv în cataloagele `var`, `bin`, `libexec`, `lib` și `include` situate în acest catalog. Dacă doriți să modificați catalogul rădăcină de instalare, folosiți opțiunea `--prefix`.

- `--localstatedir=nume_cale`

  Această opțiune modifică amplasamentul catalogului de date. O puteți folosi dacă nu doriți să vă păstrați bazele de date în catalogul `/usr/local/var`.

- `--with-low-memory`

  Fișierul sursă `sql/sql_yacc.cc` necesită o mare cantitate de memorie pentru a fi compilat, ceea ce uneori determină suprasolicitarea construcției. Printre simptomele acestei probleme se numără mesaje de eroare privind „semnalul fatal 11" sau epuizarea memoriei virtuale. Opțiunea `--with-low-memory` determină invocarea compilatorului cu opțiuni care au ca rezultat utilizarea unei cantități mai reduse de memorie.

După ce ați rulat `configure`, compilați distribuția și apoi instalați-o:

```
% make
% make install
```

Dacă nu ați folosit opțiunea `--prefix` pentru a specifica un catalog de instalare în care aveți permisiunea de scriere, este necesar să lucrați sub contul `root` pentru a rula comanda de instalare.

Dacă veți folosi numai suportul pentru clienți furnizat de distribuție și nu rulați un server, ați terminat de instalat sistemul MySQL. Dacă instalați MySQL pentru prima dată, totuși, consultați secțiunea următoare. Dacă actualizați o instalare existentă, consultați secțiunea „Pornirea serverului".

Inițializarea catalogului de date și a tabelelor de acordare

Înainte de a putea folosi sistemul dumneavoastră MySQL instalat, trebuie să inițializați baza de date `mysql`, depozit al tabelelor de acordare care controlează accesul din rețea la serverul dumneavoastră. Această operație este necesară numai pentru instalări noi și numai dacă veți rula un server. Dacă executați numai instalări ale programului client, puteți omite secțiunea de față. Pentru o distribuție binară, rulați comenzile din catalogul de instalare (părintele catalogului `bin` creat de distribuție). Pentru o distribuție sursă, rulați comenzile din catalogul amplasat la nivelul cel mai de sus al distribuției.

`DATADIR` este numele căii de acces spre catalogul dumneavoastră de date. În mod normal, veți rula comenzile următoare din contul `root`. Dacă ați deschis sesiunea de lucru ca

mysqladm sau ați instalat MySQL sub propriul dumneavoastră cont, deoarece intenționați să-l rulați personal, puteți executa comenzile fără a avea calitatea de root și puteți omite comenzile chown și chmod.

Tabelele de acordare prestabilite pot fi configurate rulând scriptul mysql_install_db:[1]

```
# scripts/mysql_install_db
```

Dacă scriptul mysql_install_db eșuează, consultați capitolul „Instalarea sistemului MySQL" din manualul de referință MySQL, pentru a vedea dacă acesta conține informații despre problema pe care o aveți, apoi reluați operația. Rețineți că dacă mysql_install_db nu se derulează complet și cu succes, toate tabelele de acordare pe care le creează sunt potențial incomplete. Trebuie să le eliminați, deoarece mysql_install_db nu va încerca să creeze din nou tabele care descoperă că există deja. Puteți elimina întreaga bază de date mysql astfel:

```
# rm -rf DATADIR/mysql
```

După ce ați rulat scriptul mysql_install_db, închideți serverul dacă instalați o versiune a sistemului MySQL mai veche decât 3.22.10 (în versiunile mai recente, mysql_install_db execută automat această operație):

```
# bin/mysqladmin --user=root shutdown
```

Serverul fiind oprit, modificați utilizatorul proprietar și grupul din care face parte acesta, precum și modul tuturor fișierelor din catalogul de date:

```
# chown -R mysqladm.mysqlgrp DATADIR
# chmod -R go-rwx DATADIR
```

Comanda chown face din utilizatorul MySQL proprietarul fișierelor respective, iar chmod modifică modul pentru a împiedica pe orice utilizator, cu excepția lui mysqladm, să obțină accesul la catalogul de date.

Pornirea serverului

Această etapă este necesară numai dacă veți rula un server. Dacă executați numai instalări ale clienților, puteți omite secțiunea de față. Rulați comenzile din această secțiune din același catalog indicat în secțiunea anterioară. În mod normal, comenzile se rulează din contul root. Dacă ați deschis sesiunea de lucru din contul mysqladm sau ați instalat MySQL sub propriul dumneavoastră cont, puteți rula comenzile fără a fi root și trebuie să omiteți opțiunea --user.

Pentru a porni serverul, folosiți următoarea comandă:

```
# bin/safe_mysqld --user=mysqladm &
```

Opțiunea --user indică serverului să ruleze sub contul mysqladm. Dacă doriți să activați caracteristica de consemnare în jurnal, folosiți în schimb această comandă:

```
# bin/safe_mysqld --user=mysqladm --log &
```

[1] Nu este necesar să executați această operație dacă instalați din fișiere RPM, deoarece mysql_install_db va fi rulat automat pentru dumneavoastră. – N.A.

Instalarea prestabilită permite utilizatorului MySQL root să se conecteze fără parolă. Stabilirea unei parole este o idee bună. De asemenea, trebuie să configurați serverul astfel încât acesta să pornească și să se oprească atunci când sistemul dumneavoastră pornește, respectiv se oprește. De asemenea, este recomandabil să se activeze caracteristica de consemnare în jurnalul de actualizare, deoarece poate fi utilă pentru refacerea datelor. Pentru instrucțiuni cu privire la executarea oricăreia dintre aceste operații, consultați capitolul 11.

Instalarea suportului pentru Perl DBI

Dacă doriți să scrieți scripturi Perl care să obțină acces la baze de date MySQL, instalați programul DBI. Acesta impune instalarea suportului MySQL pentru programarea clienților, deoarece folosește biblioteca client pentru MySQL scrisă în C. De asemenea, dacă doriți să scrieți scripturi DBI bazate pe Web, puteți instala modulul CGI.pm. Programul DBI necesită o versiune rezonabil de recentă a limbajului Perl (5.004 sau ulterioară). Dacă în sistemul dumneavoastră nu este instalat limbajul Perl, vizitați adresa http://www.perl.com/, descărcați o distribuție Perl și instalați-o înainte de a instala suportul pentru DBI.

Suportul pentru DBI necesită trei module, pe care le puteți instala din fișiere sursă sau RPM:

- **Data-Dumper.** Un modul pentru prelucrarea într-un mod convenabil a structurilor de date Perl.
- **DBI.** Principalul driver DBI.
- **Msql-Mysql-modules.** Driverul specific MySQL folosit de DBI atunci când vă conectați la un server MySQL.

Dacă instalați de la sursă, instalați modulele în ordinea prezentată în lista anterioară. În caz contrar, etapa de testare din următoarele instrucțiuni de instalare nu va funcționa.

Instalarea este similară pentru toate cele trei module. Când instalați de la sursă, despachetați un fișier de distribuție folosind una dintre următoarele comenzi (folosiți a doua comandă dacă versiunea dumneavoastră a programului tar nu înțelege opțiunea z pentru decomprimarea distribuției):

```
% tar zxf fisier_distributie.tar.gz
% gunzip < fisier_distributie.tar.gz | tar xf -
```

Apoi, deplasați-vă în catalogul de distribuție creat de comanda tar și rulați următoarele comenzi (pentru a rula etapa de instalare, aveți nevoie de contul root):

```
% perl Makefile.PL
% make
% make test
% make install
```

Dacă rulați comenzile precedente pentru distribuția Msql-Mysql-modules, comanda perl vă va pune unele întrebări atunci când generați Makefile:

- **Ce drivere doriți să instalați?** Există opțiuni pentru diferite combinații de MySQL și mSQL. Dacă nu rulați și mSQL, alegeți MySQL pentru a nu vă complica.

- **Doriți să instalați emulația MysqlPerl?** MysqlPerl este vechea interfață Perl pentru MySQL. Este depășită. Răspundeți negativ la această întrebare, cu excepția cazului când aveți scripturi MysqlPerl și doriți să activați suportul de emulație pentru acestea în modulul DBI.
- **Unde este instalat MySQL în sistemul dumneavoastră?** Acesta trebuie să fie „bunicul" catalogului care conține fișierele antet din MySQL, probabil `/usr/local` sau `/usr/local/mysql`, în afara cazului când ați instalat MySQL într-o locație non-standard.
- **Ce bază de date trebuie să folosesc pentru testarea driverelor MySQL?** Baza de date prestabilită este `test`, ceea ce nu ar trebui să ridice nici o problemă, decât dacă ați dezactivat accesul anonim la aceasta. În acest caz, trebuie să dați un nume de bază de date la care aveți acces și apoi să specificați un nume de utilizator și o parolă MySQL valabile pentru următoarele câteva întrebări.
- **Care este gazda pe care rulează baza de date?** Dacă rulați un server local, `localhost` trebuie să fie suficientă. Dacă nu, denumiți o gazdă care rulează un server la care aveți acces. Serverul MySQL trebuie să ruleze pe această gazdă atunci când rulați comanda `make test`; în caz contrar, testele vor eșua.
- **Care este numele de utilizator pentru conectarea la baza de date?**

  Care este parola pentru conectarea la baza de date?

  Numele și parola care se vor folosi pentru conectarea la serverul MySQL în vederea testării. Răspunsul prestabilit este `undef` pentru ambele întrebări. Astfel, driverul este obligat să se conecteze ca utilizator anonim. Dacă trebuie să vă conectați sub un anumit cont, specificați răspunsuri nevide.

Dacă aveți probleme la instalarea modulelor Perl, consultați fișierul `README` pentru a identifica distribuția relevantă, precum și arhivele cu mesaje pentru lista de corespondență aferentă DBI, deoarece acolo se pot găsi răspunsurile la majoritatea problemelor de instalare.

Dacă doriți să folosiți CGI.pm, poate că acesta se găsește deja în varianta dumneavoastră de instalare a limbajului Perl. Încercați să rulați comanda `perldoc CGI`. Dacă este afișată documentația CGI, atunci CGI este instalat. În caz contrar, procurați-vă distribuția, despachetați-o și instalați-o folosind aceleași instrucțiuni ca și cele pentru celelalte module Perl.

Dacă doriți să instalați modulul Apache `mod_perl` în vederea utilizării cu scripturile DBI bazate pe Web, vizitați regiunea `mod_perl` a sitului Web Apache, situată în locația următoare, pentru a obține mai multe informații:

```
http://perl.apache.org/
```

Instalarea limbajului Perl și a serverului Apache

Următoarele instrucțiuni ar trebui să-și facă efectul. Dacă întâmpinați probleme, verificați secțiunea „VERBOSE INSTALL" a fișierului `INSTALL` inclus în distribuția PHP. (Oricum, este indicat să citiți fișierul respectiv. Conține o mulțime de informații.)

Aceste instrucțiuni pornesc de la premisa că aveți la dispoziție o distribuție sursă a serverului Apache și că veți rula PHP ca modul Apache. De asemenea, se presupune că dispuneți de Apache 1.3.9 și de PHP 3.0.12, precum și că vă aflați într-un catalog în care se găsesc cataloagele `apache_1.3.9` și `php-3.0.12`, care conțin distribuțiile sursă. Dacă dispuneți de versiuni sau locații diferite, utilizați versiunile și căile de acces din sistemul dumneavoastră atunci când folosiți următoarele instrucțiuni.

Pentru a configura și construi PHP, începeți prin a rula următoarele comenzi (procedura arată puțin ciudat, deoarece începeți cu distribuția Apache, dar nu este nici o problemă):

```
% cd apache_1.3.9
% ./configure
```

Comanda `configure` pentru Apache este necesară deoarece unele dintre fișierele antet de care are nevoie PHP sunt generate la configurare. Dacă nu rulați `configure` în distribuția Apache, procesul de configurare a limbajului PHP nu va putea găsi fișierele. După ce ați configurat Apache, deplasați-vă în catalogul de distribuție PHP, configurați PHP, construiți-l și apoi instalați-l. De asemenea, puteți instala o copie a fișierului de inițializare PHP:

```
% cd ../php-3.0.12
% ./configure --with-mysql --with-apache=../apache_1.3.9 9
        --enable-track-vars
% make
% make install
% cp php3.ini-dist /usr/local/lib/php3.ini
```

Comanda `configure` pentru PHP îi indică acestuia că doriți suportul pentru MySQL și indică amplasamentul arborelui sursă Apache. Opțiunea `--enable-track-vars` activează conversia automată a datelor de intrare în formular în variabile care pot fi ușor accesibile din interiorul paginilor dumneavoastră PHP. Comenzile `make` compilează și instalează PHP. Pentru a instala fișierul de inițializare `php3.ini` aveți nevoie de contul `root`.

Când construiți PHP ca modul, „instalarea" acestuia înseamnă doar copierea modulului în arborele sursă Apache, astfel încât să poată fi legat în binarul Apache. După ce ați efectuat această operație, puteți compila și instala Apache:

```
% cd ../apache_1.3.9
% ./configure --activate-module=src/modules/php3/libphp3.a
% make
% make install
```

Aceste etape reconfigurează serverul Apache astfel încât să fie la curent cu fișierul modul PHP, apoi îl construiesc și îl instalează. Trebuie să editați fișierul de configurare Apache `httpd.conf` astfel încât Apache să recunoască scripturi PHP. Recunoașterea modulului PHP se bazează pe extensia numelor de fișier folosite pentru scripturile PHP. De exemplu, dacă doriți ca Apache să recunoască atât `.php`, cât și `.php3`, includeți următoarele linii în fișierul de configurare:

```
AddType application/x-httpd-php3 .php
AddType application/x-httpd-php3 .php3
```

Sufixele care se vor folosi depind de modul în care intenționați să vă denumiți scripturile PHP. Extensiile `.php` și `.php3` sunt, probabil, cele mai comune. O altă extensie comună

este .phtml. Le puteți pur și simplu activa pe toate, dacă doriți. Aceasta este oricum soluția cea mai indicată dacă urmează să instalați pagini pe care vi le procurați de la alte persoane, ce folosesc una sau alta dintre aceste extensii.

De asemenea, puteți cere serverului Apache să recunoască index.php sau index.php3 ca fișier prestabilit pentru un catalog, atunci când la sfârșitul unui URL nu este specificat nici un nume de fișier. Probabil că în fișierul de configurare veți găsi o linie care se prezintă astfel:

```
DirectoryIndex index.html
```

Înlocuiți-o cu următoarea:

```
DirectoryIndex index.html index.php index.php3
```

După ce ați editat fișierul de configurare Apache, opriți serverul httpd dacă acesta era deja în funcțiune și apoi porniți noul server httpd pe care tocmai l-ați instalat. Pe multe sisteme, următoarele comenzi (sub contul root) execută aceste operații:

```
# /usr/local/apache/bin/apachectl stop
# /usr/local/apache/bin/apachectl start
```

De asemenea, trebuie să configurați serverul Apache astfel încât să pornească, respectiv să se oprească la pornirea și oprirea sistemului. Vezi documentația Apache pentru instrucțiuni. În mod normal, aceasta implică rularea comenzii apachectl start la încărcarea sistemului de operare, respectiv apachectl stop la oprire.

Instalarea sistemului MySQL pe Windows

Puteți rula MySQL sub Windows 95, Windows 98 sau Windows NT. Pentru aceasta, este necesar ca suportul pentru TCP/IP să fie instalat, iar programul dumneavoastră Winsock trebuie să aibă cel puțin versiunea 2.

Sub Windows puteți instala două categorii de programe MySQL:

- Programe autonome, cum sunt cele pe care le instalați pentru UNIX (serverul mysqld și programe client, precum mysql și mysqladmin).
- MyODBC, driverul MySQL pentru ODBC, care permite altor programe (precum Access) să comunice cu serverele MySQL.

Distribuțiile Windows sunt toate disponibile din situl Web MySQL, sub formă de fișiere ZIP. Pentru a despacheta un asemenea fișier, pur și simplu executați dublu clic pe acesta. Dacă nu reușiți, folosiți un program precum Winzip sau pkunzip pentru despachetarea fișierelor. Distribuțiile primare sunt următoarele:

- mysqlwin-versiune.zip

  Distribuția completă (serverul și clienții).
- winclients-versiune.zip

  Numai programe client (mysql, mysqladmin, mysqldump și așa mai departe). Puteți folosi această distribuție dacă nu doriți să rulați serverul sub Windows.
- myodbc-versiune-win95.zip

  myodbc-versiune-nt.zip

  Suportul MyODBC pentru Windows 95 (sau Windows 98), respectiv pentru Windows NT.
- mysqlclient-versiune-cygwin-b20.tar.gz

  Clienți MySQL compilați cu setul de instrumente Cygnus. Acesta include mysqlc, o versiune a clientului mysql care dispune de o caracteristică în linie de comandă de editare a istoricului comenzilor. Dacă o instalați, va trebui să copiați fișierul bibliotecă cygwinb19.dll din C:\mysql\lib în catalogul de sistem Windows.

Instalarea unei distribuții client sau client-server

Dacă dispuneți de distribuția winclients-versiune.zip, care conține numai programe client, despachetați-o pentru a genera un dosar mysql. Mutați acest dosar în C:\.

Dacă dispuneți de distribuția mysqlwin-versiune.zip, care conține serverul și clienții, despachetați-o mai întâi pe aceasta. În dosarul rezultant, rulați programul Setup pentru a instala mysql în C:\mysql.

Puteți alege dintre următoarele servere:

Server	Descriere
mysqld	Server standard
mysqld-opt	Server optimizat pentru procesoarele Pentium
mysqld-nt	Server care poate fi instalat ca serviciu sub Windows NT

Puteți rula oricare dintre aceste servere sub NT, dar ultimul poate fi rulat numai sub NT.

Pentru a rula mysqld sau mysqld-opt, porniți serverul după cum urmează:

```
C:\> C:\mysql\bin\mysqld
C:\> C:\mysql\bin\mysqld-opt
```

Pentru a opri serverul, folosiți utilitarul mysqladmin:

```
C:\> C:\mysql\bin\mysqladmin -u root shutdown
```

Sub Windows NT, puteți rula serverul mysqld-nt ca serviciu:

```
C:\> C:\mysql\bin\mysqld-nt --install
```

Dacă porniți serverul mysqld-nt în acest mod, puteți specifica alte opțiuni inserându-le în fișierul cu opțiuni C:\my.cnf. Aceste opțiuni nu pot fi specificate în linia de comandă. Când mysqld-nt rulează ca serviciu, îl puteți porni sau opri folosind următoarele comenzi:

```
C:\> net start mysql
C:\> net stop mysql
```

De asemenea, puteți folosi utilitarul Services Control Manager, care se află în dosarul Control Panel, pentru a porni sau pentru a opri serverul, respectiv puteți face uz de utilitarul mysqladmin pentru a opri serverul de la linia de comandă:

```
C:\> C:\mysql\bin\mysqladmin -u root shutdown
```

Pentru a rula mysqld-nt ca program autonom, invocați-l astfel:

```
C:\> C:\mysql\bin\mysqld-nt --standalone
```

În acest caz, puteți specifica și alte opțiuni în linia de comandă după opțiunea --standalone, dacă doriți. Pentru a opri serverul, folosiți mysqladmin.

Dacă aveți probleme în a determina serverul să funcționeze, examinați notele referitoare la Windows din capitolul „Instalarea sistemului MySQL" din manualul de referință MySQL.

Instalarea prestabilită permite utilizatorului `root` din MySQL să se conecteze fără parolă. Pentru instrucțiuni privind stabilirea unei parole, consultați capitolul 11.

Instalarea programului MyODBC

Despachetați distribuția adecvată (există una pentru Windows 95 sau 98, respectiv o alta pentru Windows NT). În dosarul rezultant, rulați programul `Setup` pentru a instala driverul ODBC pentru MySQL. Va fi de asemenea instalat un panou de control ODBC, pe care îl puteți folosi pentru a configura driverul.

Dacă întâlniți vreo eroare, precum „Probleme la copierea fișierului MFC30.DLL", în timp ce instalați MyODBC, înseamnă că `MFC30.DLL` este folosit de o altă aplicație. În acest caz, încercați să reporniți sistemul Windows în modul sigur (safe mode) și rulați din nou programul `Setup`.

Când rulați panoul de control ODBC, veți vedea o fereastră care vă permite să stabiliți un nume al sursei de date (DSN). Executați clic pe rubrica User DSN și apoi executați clic pe butonul Add pentru a deschide o fereastră care conține driverele disponibile pentru sursele de date. Selectați driverul MySQL din listă și executați clic pe butonul Finish. Veți vedea o fereastră care vă permite să introduceți parametri de conexiune pentru sursa de date. Completați parametrii adecvați pentru conexiunea pe care doriți să o stabiliți și apoi executați clic pe butonul OK. De exemplu, pentru a configura o sursă de date pentru baza de date `samp_db` folosind serverul și contul de utilizator pe care le-am utilizat în cea mai mare parte a acestei cărți, voi completa câmpurile după cum urmează:

Numele câmpului	Valoarea câmpului
Windows DSN name (nume DSN pentru Windows)	`samp_db`
MySQL host (name or IP) (gazdă MySQL - nume sau IP)	`pit-viper.snake.net`
MySQL database name: (numele bazei de date MySQL)	`samp_db`
User (utilizator):	`paul`
Password (parolă):	`secret`

Acum trebuie să puteți folosi programe compatibile ODBC pentru a obține accesul la bazele de date MySQL.

Instalarea suportului pentru Perl DBI

Cel mai ușor lucru pe care îl aveți de făcut este să vă procurați distribuția Perl ActiveState din situl Web ActiveState (`http://www.activestate.com/`) și să o instalați. Apoi, preluați și instalați modulele Perl suplimentare de care aveți nevoie. Pentru aceasta se va folosi programul `ppm` (Perl Package Manager). Pentru a afla care sunt modulele deja instalate, folosiți această comandă:


```
C:\> C:\perl\bin\ppm info
```

Apoi, instalați modulele de care aveți nevoie folosind comenzile adecvate din următoarea listă. Probabil că CGI.pm este deja instalat, dar probabil că va trebui să instalați pachetele asociate modulului DBI:

```
C:\> C:\perl\bin\ppm install Data-Dumper
C:\> C:\perl\bin\ppm install DBI
C:\> C:\perl\bin\ppm install DBD-mysql
C:\> C:\perl\bin\ppm install CGI
```

Instalarea PHP și Apache

Sub Windows, PHP este un executabil separat, iar suportul pentru MySQL este un fișier DLL. Pentru instrucțiuni complete privind instalarea PHP pe Windows, cel mai bun lucru de făcut este să vizitați următoarea pagină Web:

```
http://www.umesd.k12.or.us/php/win32install.html
```

Obținerea și instalarea distribuției bazei de date demonstrative samp_db

Fișierele folosite pentru configurarea bazei de date demonstrative `samp_db`, precum și pentru accesul la aceasta, sunt disponibile la următoarea adresă (rețineți că adresa este sensibilă la diferența între majuscule și minuscule și trebuie tastată exact așa cum se vede mai jos):

```
http://mysql.he.net/Downloads/Contrib/Examples/
```

Regăsiți fișierul de distribuție `samp_db.tar.gz`, deplasați-vă în catalogul unde doriți să-l instalați și despachetați fișierul folosind una dintre aceste comenzi:

```
% tar zxf samp_db.tar.gz
% gunzip < samp_db.tar.gz | tar xf -
```

Când despachetați distribuția, aceasta va crea un catalog denumit `samp_db`, care conține mai multe elemente:

- Fișiere pentru crearea și încărcarea bazei de date `samp_db`. Acestea sunt folosite în capitolul 1, „Introducere în MySQL și SQL".
- Programele scrise în C din capitolul 6, „Interfața API MySQL pentru C".
- Scripturile Perl DBI și fișierul cu opțiuni `samp_db.cnf` din capitolul 7, „Interfața API pentru Perl DBI".
- Scripturile PHP și fișierul `include samp_db.inc` din capitolul 8, „Interfața API pentru PHP".

Catalogul `samp_db` conține un fișier `README` pe care trebuie să-l consultați pentru a primi instrucțiuni cu privire la utilizarea fișierelor de distribuție.

ANEXA B

Referință de tipuri de coloane

Această anexă descrie fiecare tip de coloană furnizat de MySQL. Mai multe informații despre utilizarea fiecărui tip sunt disponibile în capitolul 2, „Lucrul cu date în MySQL“. Dacă nu se specifică altceva, tipurile enumerate au fost prezente în MySQL începând cel puțin de la versiunea MySQL 3.21.0.

Specificațiile privind numele tipurilor sunt scrise folosind următoarele convenții:

- **Paranteze drepte** ([ ]). Informații opționale.
- M. Lățimea maximă de afișare; dacă nu se specifică altfel, M trebuie să fie un întreg cuprins între 1 și 255.
- D. Numărul de cifre care urmează după virgulă, pentru tipurile cu parte fracționară; D trebuie să fie un întreg cuprins între 0 și 30. De asemenea, D nu trebuie să fie mai mare decât M – 2. În caz contrar, valoarea lui M este ajustată la D + 2.

În terminologia ODBC, M și D corespund termenilor de „precizie“ și „scală“.

Pentru fiecare descriere de tip, vor fi furnizate una sau mai multe dintre următoarele categorii de informații:

- **Semnificație.** O scurtă descriere a tipului.
- **Atribute permise.** Cuvinte cheie de descriere a atributelor opționale, care pot fi asociate cu tipul coloanei în instrucțiunile CREATE TABLE sau ALTER TABLE. Atributele sunt menționate în ordine alfabetică, dar aceasta nu corespunde în mod obligatoriu cu ordinea impusă de sintaxa instrucțiunilor CREATE TABLE sau ALTER TABLE. Pentru sintaxa acestor instrucțiuni, consultați Anexa D, „Referință de sintaxă SQL“. Atributele prezentate în descrierea fiecărui tip de coloană se adaugă la atributele globale menționate mai jos.
- **Lungime permisă.** Lungimea maximă permisă a valorilor din coloane, pentru tipurile șir.
- **Interval.** Pentru tipurile numerice sau dată-oră, intervalul de valori pe care îl poate reprezenta tipul respectiv. Pentru tipuri numerice întregi, sunt date două intervale, deoarece coloanele de tip întreg pot fi cu semn sau fără semn, iar intervalele sunt diferite pentru fiecare caz în parte.
- **Valoare zero.** Pentru tipurile dată-oră, valoarea „zero“ care este stocată dacă în coloană este inserată o valoare incorectă.
- **Valoare prestabilită.** Valoarea prestabilită, dacă în specificația tipului nu există nici un atribut DEFAULT explicit.
- **Spațiu de stocare necesar.** Numărul de octeți necesar pentru stocarea valorilor din tip. Pentru unele tipuri, această valoare este fixă. Numărul de octeți variază pentru alte tipuri, în funcție de lungimea valorii stocate în coloană.

- **Comparații.** Pentru tipurile șir, arată dacă comparațiile sunt sau nu sensibile la diferența între majuscule și minuscule. Se aplică și pentru sortare și indexare, deoarece aceste operații se bazează pe comparații.
- **Sinonime.** Eventuale sinonime ale numelui tipului.
- **Note.** Alte observații referitoare la tipul respectiv.
- **Atribute globale.** Unele atribute se aplică tuturor sau aproape tuturor tipurilor de coloane; acestea vor fi prezentate aici, nu în descrierea fiecărui tip. Pentru fiecare tip pot fi specificate atributele NULL sau NOT NULL. Atributul DEFAULT valoare_prestabilită poate fi specificat pentru toate tipurile, cu excepția tipurilor BLOB și TEXT.

Tipuri numerice

MySQL furnizează tipuri numerice pentru valori întregi și cu virgulă mobilă. Tipurile pot fi selectate în funcție de domeniul de valori pe care trebuie să îl reprezentați.

Pentru tipurile întregi, o coloană trebuie să fie PRIMARY KEY sau un index UNIQUE dacă este specificat un atribut AUTO_INCREMENT. Prin inserția valorii NULL într-o coloană AUTO_INCREMENT se include o valoare cu o unitate mai mare decât valoarea curentă maximă a coloanei. Dacă este specificat atributul UNSIGNED, nu sunt permise valori negative pentru tipurile întregi.

Dacă este specificat atributul ZEROFILL, valorile pentru tipurile numerice sunt completate cu zerouri inițiale (adică scrise în stânga primei cifre a valorii – N.T.) pentru a corespunde lățimii de afișare a coloanei.

TINYINT[(*M*)]

Semnificație. Un număr întreg de dimensiuni foarte mici

Atribute permise. AUTO_INCREMENT, UNSIGNED, ZEROFILL

Interval. Între -128 și 127 (între -2^7 și 2^7-1), sau între 0 și 255 (între 0 și 2^8-1) dacă are atributul UNSIGNED

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, 0 în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 1 octet

SMALLINT[(*M*)]

Semnificație. Un număr întreg de dimensiuni mici

Atribute permise. AUTO_INCREMENT, UNSIGNED, ZEROFILL

Interval. Între -32768 și 32767 (între -2^{15} și $2^{15}-1$), sau între 0 și 65535 (între 0 și $2^{16}-1$) dacă are atributul UNSIGNED

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, 0 în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 2 octeți

MEDIUMINT[(*M*)]

Semnificație. Un număr întreg de dimensiuni medii

Atribute permise. AUTO_INCREMENT, UNSIGNED, ZEROFILL

Interval. Între -8388608 și 8388607 (între -2^{23} și 2^{23}-1), sau între 0 și 16777215 (între 0 și 2^{24}-1) dacă are atributul UNSIGNED

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, 0 în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 3 octeți

INT[(*M*)]

Semnificație. Un număr întreg de dimensiuni normale

Atribute permise. AUTO_INCREMENT, UNSIGNED, ZEROFILL

Interval. Între -2147483648 și 2147483647 (între -2^{31} și 2^{31}-1), sau între 0 și 4294967295 (între 0 și 2^{32}-1) dacă are atributul UNSIGNED

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, 0 în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 4 octeți

Sinonime. INTEGER[(M)]

BIGINT[(*M*)]

Semnificație. Un număr întreg de dimensiuni mari

Atribute permise. AUTO_INCREMENT, UNSIGNED, ZEROFILL

Interval. Între -9223372036854775808 și 9223372036854775807 (între -2^{63} și 2^{63}-1), sau între 0 și 18446744073709551615 (între 0 și 2^{64}-1) dacă are atributul UNSIGNED

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, 0 în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 8 octeți

FLOAT[(*M*,*D*)]

Semnificație. Un număr mic, cu virgulă mobilă, în simplă precizie (mai puțin precis decât DOUBLE)

Atribute permise. ZEROFILL

Interval. Valorile minime diferite de zero sunt ±1.175494351E-38; valorile maxime diferite de zero sunt ±3.402823466E+38

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, 0 în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 4 octeți

Sinonime. Anterior versiunii MySQL 3.23, FLOAT(4) este sinonim cu FLOAT cu valorile M și D prestabilite.

Notă. Începând cu MySQL 3.23, FLOAT(4) este un tip cu virgulă mobilă autentic (valorile sunt stocate la precizia integrală permisă de echipamentele dumneavoastră, nu sunt rotunjite la numărul prestabilit de cifre după virgulă).

DOUBLE[(*M*,*D*)]

Semnificație. Un număr mare, cu virgulă mobilă, în dublă precizie (mai precis decât FLOAT)

Atribute permise. ZEROFILL

Interval. Valorile minime diferite de zero sunt ±2.2250738585072014E-308, valorile maxime diferite de zero sunt ±1.7976931348623157E+308

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, 0 în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 8 octeți

Sinonime. DOUBLE PRECISION[(M,D)] și REAL[(M,D)] sunt sinonime cu DOUBLE[(M,D)]. Anterior versiunii MySQL 3.23, FLOAT(8) este sinonim cu DOUBLE cu valorile M și D prestabilite.

Notă. Începând cu MySQL 3.23, FLOAT(8) este un tip cu virgulă mobilă autentic (Valorile sunt stocate la precizia integrală permisă de echipamentele dumneavoastră, nu sunt rotunjite la numărul prestabilit de cifre după virgulă.)

DECIMAL(*M*,*D*)

Semnificație. Un număr cu virgulă mobilă, stocat sub forma unui șir (1 octet pentru fiecare cifră, virgulă sau caracter – (cratimă)).

Atribute permise. ZEROFILL

Interval. Intervalul maxim este același ca pentru DOUBLE: intervalul efectiv pentru un tip DECIMAL dat este determinat de M și D. Dacă D este 0, valorile coloanelor nu au cifre după virgulă sau parte fracționară.

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, 0 în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. M octeți pentru versiunile sistemului MySQL anterioare versiunii 3.23, M + 2 octeți pentru versiunile MySQL 3.23 și ulterioare

Sinonime. NUMERIC(M,D)

Notă. Începând de la versiunea MySQL 3.23, valoarea lui M nu include octeții necesari pentru caracterul semn sau pentru punctul zecimal, în conformitate cu ANSI SQL.

Tipuri șir

Tipurile șir din MySQL sunt frecvent folosite pentru stocarea textelor, dar au un caracter general și pot conține date arbitrare. Tipurile șir pot conține valori de lungime maximă variabilă și pot fi selectate în funcție de modul în care doriți să se facă diferența între majuscule și minuscule.

CHAR(*M*)

Semnificație. Un șir de caractere de lungime fixă, unde lungimea se poate situa între 0 și M octeți, iar M trebuie să fie un întreg cuprins între 1 și 255, anterior versiunii MySQL 3.23, respectiv între 0 și 255, după versiunea MySQL 3.23. Șirurile mai scurte de M caractere sunt completate cu spații finale la stocare. La stocare, șirurile mai lungi de M caractere sunt amputate la lungimea M. Spațiile finale (adică inserate la dreapta ultimului caracter – N.T.) sunt eliminate la regăsirea valorilor.

Atribute permise. BINARY

Lungime permisă. Între 0 și M octeți

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, "" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. M octeți

Comparații. Insensibil la diferența între majuscule și minuscule (sensibil, dacă este specificat atributul BINARY)

VARCHAR(*M*)

Semnificație. Un șir de caractere de lungime variabilă, unde lungimea se poate situa între 0 și M octeți. M trebuie să fie un întreg cuprins între 1 și 255, respectiv între 0 și 255, începând de la versiunea MySQL 3.23. Spațiile finale din cadrul valorilor sunt eliminate la stocarea acestora. La stocare, șirurile mai lungi de M caractere sunt amputate la lungimea M.

Atribute permise. BINARY

Lungime permisă. Între 0 și M octeți

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, "" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. Lungimea valorii, plus un octet pentru înregistrarea lungimii

Comparații. Insensibil la diferența între majuscule și minuscule (sensibil, dacă este specificat atributul BINARY)

TINYBLOB

Semnificație. O valoare BLOB de mici dimensiuni

Atribute permise. Nici unul, în afara atributelor globale

Lungime permisă. Între 0 și 255 (între 0 și 2^8-1) octeți

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, "" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. Lungimea valorii, plus un octet pentru înregistrarea lungimii

Comparații. Sensibil la diferența între majuscule și minuscule

BLOB

Semnificație. O valoare BLOB de dimensiuni normale

Atribute permise. Nici unul, în afara atributelor globale

Lungime permisă. Între 0 și 65535 (între 0 și 2^{16}-1) octeți

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, "" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. Lungimea valorii, plus doi octeți pentru înregistrarea lungimii

Comparații. Sensibil la diferența între majuscule și minuscule

MEDIUMBLOB

Semnificație. O valoare BLOB de dimensiuni medii

Atribute permise. Nici unul, în afara atributelor globale

Lungime permisă. Între 0 și 16777215 (între 0 și 2^{24}-1) octeți

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, "" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. Lungimea valorii, plus trei octeți pentru înregistrarea lungimii

Comparații. Sensibil la diferența între majuscule și minuscule

LONGBLOB

Semnificație. O valoare BLOB de dimensiuni mari

Atribute permise. Nici unul, în afara atributelor globale

Lungime permisă. Între 0 și 4294967295 (între 0 și 2^{32}-1) octeți

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, "" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. Lungimea valorii, plus patru octeți pentru înregistrarea lungimii

Comparații. Sensibil la diferența între majuscule și minuscule

TINYTEXT

Semnificație. O valoare TEXT de mici dimensiuni

Atribute permise. Nici unul, în afara atributelor globale

Lungime permisă. Între 0 și 255 (între 0 și 2^8-1) octeți

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, "" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. Lungimea valorii, plus un octet pentru înregistrarea lungimii

Comparații. Insensibil la diferența între majuscule și minuscule

TEXT

Semnificație. O valoare TEXT de dimensiuni normale

Atribute permise. Nici unul, în afara atributelor globale

Lungime permisă. Între 0 și 65535 (între 0 și 2^{16}-1) octeți

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, "" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. Lungimea valorii, plus doi octeți pentru înregistrarea lungimii

Comparații. Insensibil la diferența între majuscule și minuscule

MEDIUMTEXT

Semnificație. O valoare TEXT de dimensiuni medii

Atribute permise. Nici unul, în afara atributelor globale

Lungime permisă. Între 0 și 16777215 (între 0 și 2^{24}-1) octeți

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, "" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. Lungimea valorii, plus trei octeți pentru înregistrarea lungimii

Comparații. Insensibil la diferența între majuscule și minuscule

LONGTEXT

Semnificație. O valoare TEXT de dimensiuni mari

Atribute permise. Nici unul, în afara atributelor globale

Lungime permisă. Între 0 și 4294967295 (între 0 și 2^{32}-1) octeți

Valoare prestabilită. NULL dacă în coloană se acceptă valoarea NULL, "" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. Lungimea valorii, plus patru octeți pentru înregistrarea lungimii

Comparații. Insensibil la diferența între majuscule și minuscule

ENUM("valoare1","valoare2",...)

Semnificație. O enumerare; valorile coloanelor pot avea exact una dintre valorile membrilor listei

Atribute permise. Nici unul, în afara atributelor globale

Valoare prestabilită. NULL, dacă în coloană se acceptă valoarea NULL, prima valoare din enumerare în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 1 octet pentru enumerări cu 1-255 membri, 2 octeți pentru enumerări cu 256-65535 membri

Comparații. Insensibil la diferența între majuscule și minuscule (sensibil anterior versiunii MySQL 3.22.1)

SET("valoare1","valoare2",...)

Semnificație. Un set; valorile coloanelor pot avea zero sau mai multe dintre valorile membrilor listei

Atribute permise. Nici unul, în afara atributelor globale

Valoare prestabilită. NULL, dacă în coloană se acceptă valoarea NULL, "" (set vid) în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 1 octet pentru seturi cu 1-8 membri, 2 octeți (seturi cu 9-16 membri), 3 octeți (seturi cu 17-24 membri), 4 octeți (seturi cu 25-32 membri) sau 8 octeți (seturi cu 33-64 membri)

Comparații. Insensibil la diferența între majuscule și minuscule (sensibil anterior versiunii MySQL 3.22.1)

Tipuri pentru dată și oră

MySQL furnizează tipuri pentru reprezentarea datelor de tip temporal în diferite forme. Există tipuri de dată și de oră, fie împreună, fie în combinație. Există un tip special „amprentă de timp" care este actualizat automat la modificarea unei înregistrări, respectiv un tip pentru stocarea anilor, dacă nu aveți nevoie de o dată completă.

DATE

Semnificație. O dată, în format "AAAA-LL-ZZ"

Atribute permise. Nici unul, în afara atributelor globale

Interval. Între "1000-01-01" și "9999-12-31"

Valoare zero. "0000-00-00"

Valoare prestabilită. NULL, dacă în coloană se acceptă valoarea NULL, "0000-00-00" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 3 octeți (4 octeți înainte de versiunea MySQL 3.22)

TIME

Semnificație. O oră, în format "hh:mm:ss" ("-hh:mm:ss" pentru valori negative) reprezintă timpul scurs, dar poate fi tratat ca oră din zi

Atribute permise. Nici unul, în afara atributelor globale

Interval. Între "-838:59:59" și "838:59:59"

Valoare zero. "00:00:00"

Valoare prestabilită. NULL, dacă în coloană se acceptă valoarea NULL, "00:00:00" în cazul în care coloana are atributul NOT NULL

Spațiu de stocare necesar. 3 octeți

Notă. Deși este folosită ca valoare zero atunci când sunt inserate valori incorecte într-o coloană `TIME`, `"00:00:00"` este o valoare corectă, care se încadrează în domeniul normal de valori.

DATETIME

Semnificație. O dată și o oră (ambele componente sunt obligatorii), în format `"AAAA-LL-ZZ hh:mm:ss"`

Atribute permise. Nici unul, în afara atributelor globale

Interval. Între `"1000-01-01 00:00:00"` și `"9999-12-31 23:59:59"`

Valoare zero. `"0000-00-00 00:00:00"`

Valoare prestabilită. `NULL`, dacă în coloană se acceptă valoarea `NULL`, `"0000-00-00 00:00:00"` în cazul în care coloana are atributul `NOT NULL`

Spațiu de stocare necesar. 8 octeți

TIMESTAMP[(*M*)]

Semnificație. O amprentă de timp (dată și oră), în format `"AAAALLZZhhmmss"`

Atribute permise. Nici unul, în afara atributelor globale

Interval. Între `19700101000000` și undeva în anul 2037

Valoare zero. `00000000000000`

Valoare prestabilită. Data și ora curentă. Rețineți că `DESCRIBE` și `SHOW COLUMNS` raportează valoarea prestabilită ca `NULL`

Spațiu de stocare necesar. 4 octeți

Notă. Prin inserția unei valori `NULL` în prima coloană `TIMESTAMP` a unui tabel se inserează data și ora curentă. Modificarea valorii oricărei alte coloane din rând determină actualizarea primei coloane `TIMESTAMP` cu data și ora modificării. Valorile sunt stocate și folosite intern în calcule la precizie completă cu 14 caractere, indiferent de lățimea de afișare. Dacă este specificat `NOT NULL` ca atribut, acesta va fi ignorat.

YEAR

Semnificație. Un an, în format `AAAA`

Atribute permise. Nici unul, în afara atributelor globale

Interval. Între `1900` și `2155`

Valoare zero. `0000`

Valoare prestabilită. `NULL`, dacă în coloană se acceptă valoarea `NULL`, `0000` în cazul în care coloana are atributul `NOT NULL`

Spațiu de stocare necesar. 1 octet

Notă. Tipul de coloană `YEAR` a fost introdus în MySQL 3.22.

ANEXA C

Referință de operatori și funcții

Această anexă enumeră toate funcțiile MySQL pe care le puteți folosi în instrucțiuni SQL. De asemenea, enumeră operatorii folosiți pentru construirea expresiilor. Evaluarea unei expresii implică deseori conversii de tip ale valorilor din expresia respectivă. Consultați capitolul 2, „Lucrul cu date în MySQL", pentru detalii privind circumstanțele în care se produce conversia de tip și regulile folosite de MySQL pentru conversia valorilor de la un tip la altul.

Dacă nu este specificat altceva, operatorii și funcțiile prezentate aici sunt prezente în MySQL începând cel puțin de la versiunea 3.21.0. Dacă un operator sau funcție se comportă într-un mod diferit față de cel prezentat aici, examinați notele de modificare din manualul de referință MySQL. Este foarte posibil ca versiunile mai vechi de MySQL să conțină hibe care au fost remediate deja.

Exemplele de operatori și funcții sunt scrise în formatul următor:

```
expresie                                   ⇨ rezultat
```

`expresie` ilustrează modul de utilizare a unui operator sau a unei funcții, iar `rezulta` indică rezultatul obținut din evaluarea expresiei. De exemplu:

```
LOWER ("ABC")                              ⇨ "abc"
```

Aceasta înseamnă că apelul la funcția `LOWER("ABC")` produce rezultatul șir `"abc"`.

Puteți verifica personal exemplele prezentate în această anexă, folosind programul `mysql`. Pentru a încerca exemplul precedent, invocați programul `mysql`, tastați expresia dată ca exemplu, precedată de `SELECT` și urmată de un caracter punct și virgulă, după care apăsați pe tasta Enter:

```
mysql> SELECT LOWER("ABC");
+--------------+
| LOWER("ABC") |
+--------------+
| abc          |
+--------------+
```

MySQL nu impune ca o instrucțiune `SELECT` să aibă o clauză `FROM`, ceea ce facilitează exersarea utilizării operatorilor și a funcțiilor, introducând expresii arbitrare în acest mod. (Unele sisteme de baze de date nu vă permit să emiteți o instrucțiune `SELECT` fără clauza `FROM` – nefericită restricție!)

Exemplele includ instrucțiuni `SELECT` complete pentru funcții a căror utilizare nu poate fi demonstrată altfel. Secțiunea „Funcții de sumar" este scrisă în acest mod, deoarece funcțiile respective nu au sens decât dacă se face referire la un anumit tabel.

Numele funcțiilor, precum și operatorii de tip cuvânt, precum BETWEEN, pot fi specificate folosind atât majuscule, cât și minuscule.

Anumite tipuri de argumente ale funcțiilor apar în mod repetat și sunt reprezentate de nume cu o semnificație fixă:

- *expr* reprezintă o expresie; în funcție de context, aceasta poate fi o expresie numerică, șir, de tip dată sau oră, și poate încorpora constante, referințe la coloane de tabel sau alte expresii.
- *sir* reprezintă un șir; poate fi un șir literal, o referință la o coloană dintr-un tabel cu valori de tip șir sau o expresie care produce un șir.
- *n* reprezintă un întreg (ca și literele situate lângă n în alfabet).
- *x* reprezintă un număr cu virgulă mobilă (ca și literele plasate lângă x în alfabet).

Alte nume de argumente sunt folosite mai puțin frecvent și sunt definite în locul de utilizare. Componentele opționale ale secvențelor de operatori sau de apeluri la funcții sunt indicate prin paranteze drepte ([ ]).

Operatori

Operatorii sunt folosiți pentru a combina termenii din expresii pentru a executa operații aritmetice, pentru compararea valorilor, pentru a executa operații la nivel de bit sau logice, respectiv pentru a identifica echivalențele cu un model.

Precedența operatorilor

Operatorii au niveluri de precedență variabile. Nivelurile sunt prezentate în lista următoare, în ordine descrescătoare. Operatorii situați pe aceeași linie au aceeași precedență. Operatorii de la un nivel de precedență dat sunt evaluați de la stânga la dreapta. Operatorii de la un nivel de precedență mai ridicat sunt evaluați înaintea operatorilor de la un nivel de precedență mai redus.

```
BINARY
NOT !
- (minus unar)
* / %
+ -
<< >>
&
|
< <= = <=> != <> >= > IN IS LIKE REGEXP RLIKE
BETWEEN
AND &&
OR ||
```

Operatorii unari (minus unar, NOT și BINARY) se asociază mai strâns decât operatorii binari. Cu alte cuvinte, aceștia se grupează cu termenul imediat următor dintr-o expresie, nu cu restul expresiei în totalitatea sa.


```
-2+3                    ⇨ 1
-(2+3)                  ⇨ -5
```

Gruparea operatorilor

Parantezele rotunde de deschidere, respectiv închidere, se pot folosi pentru a grupa părți ale unei expresii, cu scopul de a anula precedența prestabilită a operatorilor, care determină ordinea în care sunt evaluați termenii unei expresii (vezi „Precedența operatorilor"). De asemenea, parantezele mai pot fi folosite pur și simplu pentru a asigura claritatea vizuală, pentru a facilita citirea unei expresii.

```
1 + 2 * 3 / 4           ⇨ 2.50
(((1 + 2) * 3) / 4)     ⇨ 2.25
```

Operatori aritmetici

Acești operatori execută operații aritmetice standard. Operatorii aritmetici lucrează cu numere, nu șiruri (deși șirurile care seamănă cu numerele sunt convertite automat la valoarea numerică adecvată). Operațiile aritmetice cu valori NULL produc rezultatul NULL.

- +

  Calculează suma argumentelor.

```
2 + 2                   ⇨ 4
3.2 + 4.7               ⇨ 7.9
"43bc" + "21d"          ⇨ 64
"abc" + "def"           ⇨ 0
```

  Exemplul final din această listă arată că operatorul de adunare nu servește ca operator de concatenare a șirurilor, ca în alte limbaje. În schimb, șirurile sunt convertite în numere înainte ca operația aritmetică să aibă loc. Șirurile care nu seamănă cu numere sunt convertite la 0. Pentru a concatena șirurile, folosiți funcția CONCAT().

- -

  Calculează diferența dintre operanzi, atunci când este folosit între doi termeni ai unei expresii. Determină negativul operandului, atunci când este folosit în fața unui singur termen (adică schimbă semnul termenului).

```
10 - 7                  ⇨ 3
- (10 - 7)              ⇨ -3
```

- *

  Calculează produsul operanzilor.

```
2 * 3                   ⇨ 6
2.3 * -4.5              ⇨ -10.3
```

- /

  Calculează câtul a doi operanzi. Împărțirea cu zero produce un rezultat NULL.

```
3 / 1                   ⇨ 3.00
1 / 3                   ⇨ 0.33
1 / 0                   ⇨ NULL
```

- %

  Operatorul modulo; calculează restul împărțirii lui m la n. m % n este similar cu MOD(m,n). Ca în cazul împărțirii, operatorul modulo cu un divizor egal cu zero returnează NULL.

```
12 % 4        ⇨ 0
12 % 5        ⇨ 2
12 % 0        ⇨ NULL
```

Pentru operatorii de adunare, scădere și înmulțire, operațiile aritmetice sunt executate cu valori BIGINT (întregi pe 64 de biți) dacă ambele argumente sunt întregi. Aceasta înseamnă că expresii cu valori mari pot depăși domeniul calculelor cu numere întregi pe 64 de biți, putând duce la rezultate imprevizibile:

```
999999999999999999 * 999999999999999999      ⇨ -7527149226598858751
99999999999 * 99999999999 * 99999999999      ⇨ -1504485813132150785
18014398509481984 * 18014398509481984        ⇨ 0
```

Pentru operatorii de împărțire și modulo, valorile BIGINT sunt folosite numai atunci când împărțirea se efectuează într-un context unde rezultatul este convertit la un întreg.

Operatori de comparație

Operatorii de comparație returnează 1 în cazul în care comparația este adevărată, respectiv 0 în caz contrar. Puteți compara numere sau șiruri. Operanzii sunt convertiți după necesități, în conformitate cu următoarele reguli:

- Cu excepția operatorului <=>, comparațiile care implică valori NULL determină rezultatul NULL. (Operatorul <=> este similar cu =, cu excepția faptului că NULL <=> NULL este o afirmație adevărată.)
- Dacă ambii operanzi sunt șiruri, atunci vor fi comparați din punct de vedere lexicografic, ca șiruri. Comparațiile între șiruri se execută folosind setul de caractere în vigoare pe server.
- Dacă ambii operanzi sunt numere întregi, atunci sunt comparați din punct de vedere numeric, ca întregi.
- Constantele hexazecimale care nu sunt comparate cu un număr sunt comparate sub formă de șiruri binare.
- Dacă unul din operanzi este o valoare TIMESTAMP sau DATETIME și celălalt este o constantă, operanzii sunt comparați sub formă de valori TIMESTAMP. Această operație are drept scop o mai bună comportare a comparațiilor pentru aplicațiile ODBC.
- Dacă nu se aplică nici una dintre regulile precedente, operanzii sunt comparați numeric, ca valori cu virgulă mobilă. Rețineți că aici este inclusă și situația în care se compară un șir cu un număr. Șirul este convertit într-un număr, ceea ce are ca rezultat o valoare zero dacă șirul nu seamănă cu un număr. De exemplu, "14.3" este convertit la 14.3, dar "L4.3" este convertit la zero.

Comparațiile următoare ilustrează regulile de mai sus:

```
2 < 12          ⇨ 1
"2" < "12"      ⇨ 0
"2" < 12        ⇨ 1
```


Prima comparație implică doi întregi, care sunt comparați ca numere. Cea de-a doua comparație implică două șiruri, care sunt comparate din punct de vedere lexicografic. Cea de-a treia comparație implică un șir și un număr, deci vor fi comparate ca valori cu virgulă mobilă.

Comparațiile între șiruri nu sunt sensibile la diferența între majuscule și minuscule decât în cazul în care comparația implică un șir binar. Astfel, se execută o comparație sensibilă la diferența între majuscule și minuscule dacă folosiți cuvântul cheie BINARY sau dacă executați comparații între valori din coloane CHAR BINARY, VARCHAR BINARY sau BLOB.

- =

  Are ca rezultat 1 dacă operanzii sunt egali, respectiv 0 în caz contrar.

```
1 = 1                     ⇨ 1
1 = 2                     ⇨ 0
"abc" = "abc"             ⇨ 1
"abc" = "def"             ⇨ 0
"abc" = "ABC"             ⇨ 1
BINARY "abc" = "ABC"      ⇨ 0
BINARY "abc" = "abc"      ⇨ 1
"abc" = 0                 ⇨ 1
```

  "abc" este egal atât cu "abc", cât și cu "ABC", deoarece în mod prestabilit comparațiile între șiruri nu sunt sensibile la diferența între majuscule și minuscule. Comparațiile dintre șiruri pot căpăta această sensibilitate dacă se folosește operatorul BINARY. "abc" este egal cu 0 deoarece este convertit la un număr, în conformitate cu regulile de comparație. Deoarece "abc" nu arată ca un număr, este convertit la 0.

- <=>

  Operator de egalitate care acceptă și valori NULL; este similar cu operatorul de egalitate, cu excepția faptului că determină valoarea 1 când operanzii sunt egali, chiar și atunci când au valoarea NULL.

```
1 <=> 1          ⇨ 1
1 <=> 2          ⇨ 0
NULL <=> NULL    ⇨ 1
NULL = NULL      ⇨ NULL
```

  Ultimele două exemple prezintă modul diferit în care operatorii = și <=> tratează comparațiile cu operatori NULL.

  Operatorul <=> a fost introdus în versiunea MySQL 3.23.0.

- != sau <>

  Determină rezultatul 1 dacă operanzii sunt inegali, respectiv 0 în caz contrar.

```
3.4 != 3.4               ⇨ 0
"abc" <> "ABC"           ⇨ 0
BINARY "abc" <> "ABC"    ⇨ 1
"abc" != "def"           ⇨ 1
```

- `<`

  Determină rezultatul 1 dacă operandul din stânga este mai mic decât operandul din dreapta; în caz contrar, rezultatul este 0.

```
3 < 10                  ⇨ 1
105.4 < 10e+1           ⇨ 0
"abc" < "ABC"           ⇨ 0
"abc" < "def"           ⇨ 1
```

- `<=`

  Determină rezultatul 1 dacă operandul din stânga este mai mic sau egal decât operandul din dreapta; în caz contrar, rezultatul este 0.

```
"abc" <= "a"            ⇨ 0
"a" <= "abc"            ⇨ 1
13.5 <= 14              ⇨ 1
(3 * 4) - (6 * 2) <= 0  ⇨ 1
```

- `>=`

  Determină rezultatul 1 dacă operandul din stânga este mai mare sau egal decât operandul din dreapta; în caz contrar, rezultatul este 0.

```
"abc" >= "a"            ⇨ 1
"a" >= "abc"            ⇨ 0
13.5 >= 14              ⇨ 0
(3 * 4) - (6 * 2) >= 0  ⇨ 1
```

- `>`

  Determină rezultatul 1 dacă operandul din stânga este mai mare decât operandul din dreapta; în caz contrar, rezultatul este 0.

```
PI() > 3                ⇨ 1
"abc" > "a"             ⇨ 1
SIN(0) > COS(0)         ⇨ 0
```

- *expr* `BETWEEN` *min* `AND` *max*

  Determină rezultatul 1 dacă `min` este mai mic sau egal cu `expr` și dacă `max` este mai mare sau egal cu `expr`. Dacă operanzii `expr`, `min` și `max` sunt toți de același tip, aceste expresii sunt echivalente:

```
expr BETWEEN min AND max
(min <= expr AND expr <= max)
```

  Dacă operanzii nu sunt de același tip, se va produce o conversie de tip și este posibil ca aceste expresii să nu fie echivalente. `BETWEEN` este evaluat folosind comparații determinate în conformitate cu tipul operandului `expr`:

  - Dacă *expr* este un șir, operanzii sunt comparați lexicografic, ca șiruri. Comparațiile sunt sau nu sensibile la diferența între majuscule și minuscule, dacă `expr` este sau nu un șir binar.
  - Dacă *expr* este un întreg, operanzii sunt comparați numeric, ca întregi.
  - Dacă nici una dintre regulile anterioare nu este adevărată, operanzii sunt comparați numeric, sub formă de numere cu virgulă mobilă.


```
"def" BETWEEN "abc" AND "ghi"       ⇨ 1
"def" BETWEEN "abc" AND "def"       ⇨ 1
13.3 BETWEEN 10 AND 20              ⇨ 1
13.3 BETWEEN 10 AND 13              ⇨ 0
2 BETWEEN 2 AND 2                   ⇨ 1
B BETWEEN "A" AND "a"               ⇨ 0
BINARY B BETWEEN "A" AND "a"        ⇨ 1
```

  `BETWEEN` a fost introdus în MySQL versiunea 3.21.2.

- *expr* `IN (`*valoare1*`,`*valoare2*`,...)`

  expr `NOT IN (`*valoare1*`,`*valoare2*`,...)`

  `IN` determină rezultatul 1 dacă `expr` este una dintre valorile din listă, respectiv 0 în caz contrar. Pentru `NOT IN`, situația se inversează. Expresiile următoare sunt echivalente:

```
expr NOT IN (valoare1,valoare2,...)
NOT (expr IN (valoare1,valoare2,...))
```

  Dacă toate valorile din listă sunt constante, MySQL le sortează și determină rezultatul testului `IN` folosind o căutare binară, care este foarte rapidă.

```
3 IN (1,2,3,4,5)                    ⇨ 1
"d" IN ("a","b","c","d","e")        ⇨ 1
"f" IN ("a","b","c","d","e")        ⇨ 0
3 NOT IN (1,2,3,4,5)                ⇨ 0
"d" NOT IN ("a","b","c","d","e")    ⇨ 0
"f" NOT IN ("a","b","c","d","e")    ⇨ 1
```

  Operatorul `IN` a fost introdus în MySQL versiunea 3.21.0.

- *expr* `IS NULL`

  expr `IS NOT NULL`

  `IS NULL` determină rezultatul 1 dacă valoarea operandului `expr` este `NULL`, respectiv 0 în caz contrar. `IS NOT NULL` determină rezultatul invers. Expresiile următoare sunt echivalente:

```
expr IS NOT NULL
NOT (expr IS NULL)
```

  `IS NULL` și `IS NOT NULL` trebuie utilizate pentru a determina dacă valoarea operandului `expr` este sau nu `NULL`. Pentru aceasta, nu puteți folosi operatorii de comparație obișnuiți, precum `=` și `!=`.

```
NULL IS NULL                        ⇨ 1
0 IS NULL                           ⇨ 0
NULL IS NOT NULL                    ⇨ 0
0 IS NOT NULL                       ⇨ 1
NOT (0 IS NULL)                     ⇨ 1
NOT (NULL IS NULL)                  ⇨ 0
NOT NULL IS NULL                    ⇨ 1
```

  Ultimul exemplu returnează rezultatul respectiv deoarece `NOT` se asociază mai strâns decât `IS` (vezi „Precedența operatorilor").

Operatori pe bit

Operațiile pe bit sunt efectuate folosind valori BIGINT (întregi pe 64 de biți), care limitează domeniul maxim al operațiilor. Operațiile pe bit care implică valorile NULL determină un rezultat NULL.

- |

  Evaluează rezultatul operației SAU (reuniune) la nivel de bit a operanzilor.

```
1 | 1                ⇨ 1
1 | 2                ⇨ 3
1 | 2| 4 | 8         ⇨ 15
1 | 2| 4 | 8 | 15    ⇨ 15
```

- &

  Evaluează rezultatul operației ȘI (intersecție) la nivel de bit a operanzilor.

```
1 & 1                ⇨ 1
1 & 2                ⇨ 0
7 & 5                ⇨ 5
```

- <<

  Deplasează operandul stâng spre stânga, cu un număr de poziții de bit indicate de operandul din dreapta. Deplasarea cu o cantitate negativă determină rezultatul zero.

```
1 << 2               ⇨ 4
2 << 2               ⇨ 8
1 << 62              ⇨ 4611686018427387904
1 << 63              ⇨ -9223372036854775808
1 << 64              ⇨ 0
```

  Ultimele două exemple ilustrează limitele calculelor pe 64 de biți.

  Operatorul << a fost introdus în MySQL versiunea 3.22.2.

- >>

  Deplasează operandul stâng spre dreapta, cu un număr de poziții de bit indicate de operandul din dreapta. Deplasarea cu o cantitate negativă determină rezultatul zero.

```
16 >> 3              ⇨ 2
16 >> 4              ⇨ 1
16 >> 5              ⇨ 0
```

  Operatorul >> a fost introdus în MySQL versiunea 3.22.2.

Operatori logici

Operatorii logici (cunoscuți și sub numele de *operatori booleeni*, de la numele matematicianului George Boole, care a formalizat utilizarea acestora) testează adevărul sau falsitatea expresiilor. Toate operațiile logice returnează 1 pentru „adevărat", respectiv 0 pentru „fals". Operatorii logici interpretează operanzii diferiți de zero ca „adevărat", respectiv operanzii egali cu 0 ca „fals". Valorile NULL sunt manipulate conform indicațiilor aferente operatorilor.

Operatorii logici se așteaptă ca operanzii să fie numere, deci operanzii de tip șir sunt convertiți în numere înainte de evaluarea operatorului.

- NOT sau !

  Negație logică; determină rezultatul 1 dacă operandul următor este FALSE, respectiv 0 dacă operandul are valoarea TRUE, cu o singură excepție: NOT NULL este NULL.

```
NOT 0                ⇨ 1
NOT 1                ⇨ 0
NOT NULL             ⇨ NULL
NOT 3                ⇨ 0
NOT NOT 1            ⇨ 1
NOT "1"              ⇨ 0
NOT "0"              ⇨ 1
NOT ""               ⇨ 1
NOT "abc"            ⇨ 1
```

- OR sau ||

  SAU logic; determină rezultatul 1 dacă oricare dintre operanzi este adevărat (diferit de zero sau NULL), respectiv 0 în caz contrar.

```
0 OR 0               ⇨ 0
0 OR 3               ⇨ 1
4 OR 2               ⇨ 1
1 OR NULL            ⇨ 1
```

- AND sau &&

  ȘI logic; determină rezultatul 1 dacă ambii operanzi sunt adevărați (diferiți de zero sau NULL), respectiv 0 în caz contrar.

```
0 AND 0              ⇨ 0
0 AND 3              ⇨ 0
4 AND 2              ⇨ 1
1 AND NULL           ⇨ 0
```

În MySQL, operatorii !, || și && indică operații logice, așa cum indică în C. Rețineți mai ales că operatorul || nu execută concatenarea șirurilor, așa cum procedează în unele versiuni ale sistemului SQL. Pentru concatenarea șirurilor, folosiți în schimb funcția CONCAT().

Operatori de conversie

BINARY determină tratarea operatorului imediat următor ca șir binar, astfel încât comparațiile care implică șirul devin sensibile la diferența între majuscule și minuscule. Dacă operandul următor este un număr, va fi convertit la șir:

```
"abc" = "ABC"            ⇨ 1
"abc" = BINARY "ABC"     ⇨ 0
BINARY "abc" = "ABC"     ⇨ 0
"2" < 12                 ⇨ 1
"2" < BINARY 12          ⇨ 0
```

BINARY determină o conversie de la număr la șir; apoi, comparația este executată după criterii lexicografice, deoarece ambii operanzi sunt șiruri.

BINARY a fost introdus în MySQL versiunea 3.23.0.

Operatori de determinare a echivalențelor cu un model

MySQL furnizează caracteristica de determinare a echivalențelor cu un model folosind LIKE și echivalența cu modele compuse din expresii regulate extinse folosind REGEXP. Echivalențele cu modelele SQL nu sunt sensibile la diferența între majuscule și minuscule, cu excepția situațiilor când șirul care trebuie găsit sau șirul mode sunt șiruri binare. Echivalențele cu modele compuse din expresii regulate extinse sunt întotdeauna sensibile la diferența între majuscule și minuscule.

Echivalența cu un model SQL reușește numai dacă modelul corespunde cu întregul șir care trebuie găsit. Echivalențele cu modele compuse din expresii regulate extinse reușesc dacă modelul se găsește oriunde în interiorul șirului.

- *sir* LIKE *mod* [ESCAPE 'c']

  sir NOT LIKE *mod* [ESCAPE 'c']

  LIKE execută o operație simplă SQL de stabilire a echivalențelor cu un model și determină valoarea 1, dacă șirul model mod corespunde întregii expresii șir sir. Dacă nu corespunde, rezultatul este 0. Pentru NOT LIKE, rezultatele se inversează. Aceste două expresii sunt echivalente:

  ```
  sir NOT LIKE mod [ESCAPE 'c']
  NOT (sir LIKE mod [ESCAPE 'c'])
  ```

  Dacă oricare dintre șiruri este NULL, rezultatul este NULL.

  În modelele SQL, două caractere au o semnificație specială și servesc drept caractere de înlocuire, așa cum se poate vedea mai jos:

Caracter	Semnificație
%	Corespunde oricărei secvențe de caractere alta decât NULL (inclusiv șirul vid)
_	Corespunde oricărui caracter individual

  Modelele pot conține unul sau ambele caractere de înlocuire:

  ```
  "catnip" LIKE "cat%"         ⇨ 1
  "dogwood" LIKE "%wood"       ⇨ 1
  "bird" LIKE "____"           ⇨ 1
  "bird" LIKE "___"            ⇨ 0
  "dogwood" LIKE "%wo__"       ⇨ 1
  ```

  Sensibilitatea la diferența între majuscule și minuscule a operației de stabilire a echivalențelor cu un model în SQL folosind operatorul LIKE este determinată de șirurile care se compară. În mod normal, comparațiile nu sunt sensibile la diferența între majuscule și minuscule. Dacă oricare dintre șiruri este un șir binar, comparația prezintă sensibilitatea respectivă:


  ```
  "abc" LIKE "ABC"             ⇨ 1
  BINARY "abc" LIKE "ABC"      ⇨ 0
  "abc" LIKE BINARY "ABC"      ⇨ 0
  ```

  Deoarece caracterul % corespunde oricărei secvențe de caractere, corespunde chiar și absenței oricărui caracter:

  ```
  "" LIKE "%"                  ⇨ 1
  "cat" LIKE "cat%"            ⇨ 1
  ```

  În MySQL, puteți folosi LIKE cu expresii numerice:

  ```
  50 + 50 LIKE "1%"            ⇨ 1
  200 LIKE "2__"               ⇨ 1
  ```

  Pentru a stabili o corespondență literală cu un caracter de înlocuire, dezactivați semnificația specială a acestuia precedându-l cu un caracter escape, și anume \:

  ```
  "100% pur" LIKE "100%"       ⇨ 1
  "100% pur" LIKE "100\%"      ⇨ 0
  "100% pur" LIKE "100\% pur"  ⇨ 1
  ```

  Dacă doriți să specificați un caracter escape altul decât \, specificați-l folosind o clauză ESCAPE:

  ```
  "100% pur" LIKE "100^%" ESCAPE '^'      ⇨ 0
  "100% pur" LIKE "100^% pur" ESCAPE '^'  ⇨ 1
  ```

- *sir* REGEXP *mod*

  sir NOT REGEXP *mod*

  REGEXP stabilește echivalențe cu modele compuse din expresii regulate extinse. Expresiile regulate extinse sunt similare cu modelele folosite de utilitarele UNIX grep ș sed. Secvențele de modele pe care le puteți folosi sunt prezentate în tabelul C.1.

  REGEXP determină un rezultat egal cu 1 dacă șirul model mod corespunde expresiei șir sir, respectiv 0 în caz contrar. NOT REGEXP determină rezultate inverse. Următoarel două expresii sunt echivalente:

  ```
  sir NOT REGEXP mod
  NOT (sir REGEXP mod)
  ```

  Dacă oricare dintre șiruri este NULL, rezultatul este NULL.

  Modelul nu trebuie să corespundă întregului șir; este suficient să se găsească undev în interiorul șirului.

  Construcțiile [...] și [^...] reprezintă clase de caractere. În cadrul unei clase, u domeniu de caractere se poate indica folosind o liniuță între cele două caracter extreme ale domeniului. De exemplu [a-z] corespunde oricărei litere minuscule, ia [0-9] corespunde oricărei cifre. Pentru a putea indica în cadrul unei clase un caracte literal de tip paranteză dreaptă de închidere, acesta trebuie să fie primul caracter al cla sei. Pentru a putea indica un caracter literal de tip cratimă, acesta trebuie să fie primu sau ultimul caracter al clasei. De asemenea, pentru a putea indica un caracter ^, acest nu trebuie să fie primul caracter după paranteza dreaptă de deschidere.

Tabelul C.1 Secvențe de expresii regulate extinse

Secvență	Semnificație
^	Corespunde începutului șirului
$	Corespunde sfârșitului șirului
.	Corespunde oricărui caracter individual, inclusiv caracterului linie nouă
[...]	Corespunde oricărui caracter care apare între paranteze
[^...]	Corespunde oricărui caracter care nu apare între paranteze
*e**	Corespunde unui număr de zero sau mai multe instanțe ale elementului de model *e*
e+	Corespunde unui număr de una sau mai multe instanțe ale elementului de model *e*
e?	Corespunde unui număr de 0-1 instanțe ale elementului de model *e*
e1|*e2*	Corespunde elementului de model *e1* sau *e2*
e{*m*}	Corespunde unui număr de *m* instanțe ale elementului de model *e*
e{*m*,}	Corespunde unui număr de *m* sau mai multe instanțe ale elementului de model *e*
e{,*n*}	Corespunde unui număr de instanțe ale elementului de model *e* cuprins între 0 și *n*
e{*m*,*n*}	Corespunde unui număr de instanțe ale elementului de model *e* cuprins între *m* și *n*
(...)	Grupează elementele de model într-un singur element
altele	Caracterele non-speciale corespund lor înșile

În cadrul unei clase de caractere se pot folosi și numeroase alte construcții speciale mai complicate, legate de colaționarea secvențelor și de clasele de echivalență. Pentru mai multe informații, examinați manualul de referință MySQL.

O echivalență cu o expresie regulată extinsă este stabilită dacă modelul se găsește oriunde în șirul căutat, dar puteți folosi caracterele ^ și $ pentru a forța găsirea modelului numai la începutul sau la sfârșitul șirului.

```
"abcde" REGEXP "b"        ⇨ 1
"abcde" REGEXP "^b"       ⇨ 0
"abcde" REGEXP "b$"       ⇨ 0
"abcde" REGEXP "^a"       ⇨ 1
"abcde" REGEXP "e$"       ⇨ 1
"abcde" REGEXP "^a.*e$"   ⇨ 1
```

Echivalențele cu expresiile regulate extinse sunt sensibile la diferența între majuscule și minuscule:

```
"abc" REGEXP "ABC"        ⇨ 0
"ABC" REGEXP "ABC"        ⇨ 1
```

- *sir* RLIKE *mod*

  sir NOT RLIKE *mod*

  RLIKE și NOT RLIKE sunt asemănătoare cu REGEXP, respectiv NOT REGEXP.

MySQL folosește o sintaxă similară cu C pentru secvențele escape din cadrul șirurilor. De exemplu \n, \t și \\ sunt interpretate drept caracter linie nouă, tabulator și backslash. Pentru a specifica asemenea caractere într-un model, dublați backslash-urile (\\n, \\t și \\\\). Un backslash este eliminat la analiza interogării, iar secvența escape rămasă este interpretată la identificarea echivalențelor cu modelul.

Funcții

Funcțiile sunt apelate pentru a efectua un calcul și pentru a returna o valoare. Funcțiile trebuie să fie invocate fără spații între numele funcției și paranteza care urmează după acesta:

`NOW()`	Corect
`NOW ()`	Incorect

Dacă transferați unei funcții mai multe argumente, argumentele trebuie separate prin virgulă. Este permisă inserția de spații în jurul argumentelor funcțiilor:

`CONCAT("abc","def")`	Corect
`CONCAT( "abc" , "def" )`	Și așa este corect

Funcții de comparație

- GREATEST(*expr1*,*expr2*,...)

  Returnează argumentul cel mai mare, unde noțiunea de „cel mai mare" este definită conform următoarelor reguli:

  - Dacă funcția este apelată într-un context de întregi sau toate argumentele sunt întregi, argumentele sunt comparate ca numere întregi.
  - Dacă funcția este apelată într-un context de valori cu virgulă mobilă sau toate argumentele sunt valori cu virgulă mobilă, argumentele sunt comparate ca valori cu virgulă mobilă.
  - În cazul în care cele două reguli precedente nu sunt adevărate, argumentele sunt comparate sub formă de șiruri. Comparațiile nu sunt sensibile la diferența între majuscule și minuscule decât dacă unul din argumente este un șir binar.

  ```
  GREATEST(2,3,1)                  ⇨ 3
  GREATEST(38.5,94.2,-1)           ⇨ 94.2
  GREATEST("a","ab","abc")         ⇨ "abc"
  GREATEST(1,3,5)                  ⇨ 5
  GREATEST("A","b","C")            ⇨ "C"
  GREATEST(BINARY "A","b","C")     ⇨ "b"
  ```

  Funcția GREATEST() a fost introdusă în versiunea 3.22.5. În versiunile anterioare, puteți folosi în schimb funcția MAX().

- IF(*expr1*,*expr2*,*expr3*)

  Dacă expr1 este adevărată (nu este 0 sau NULL), returnează expr2; în caz contrar, returnează expr3. IF() returnează un număr sau un șir, conform contextului în care este folosită funcția.

```
IF(1,"true","false")                  ⇨ "true"
IF(0,"true","false")                  ⇨ "false"
IF(NULL,"true","false")               ⇨ "false"
IF(1.3,"non-zero","zero")             ⇨ "non-zero"
IF(0.3,"non-zero","zero")             ⇨ "zero"
IF(0.3 != 0,"non-zero","zero")        ⇨ "non-zero"
```

expr este evaluată ca o valoare întreagă, iar ultimele trei exemple indică modul în care puteți fi păcălit de această comportare, dacă nu sunteți atent. 1.3 se convertește la valoarea întreagă 1, care este adevărată. Dar 0.3 se convertește la valoarea întreagă 0, care este falsă. Ultimul exemplu din listing prezintă modalitatea adecvată de utilizare a unui număr cu virgulă mobilă: testați-l folosind o comparație, întrucât comparația returnează 1 sau 0, în funcție de rezultatul testului.

- `IFNULL(expr1,expr2)`

  Returnează *expr2* dacă valoarea expresiei *expr1* este `NULL`; în caz contrar, returnează `expr1`. `IFNULL()` returnează un număr sau un șir, conform contextului în care este folosită funcția.

```
IFNULL(NULL, "null")                  ⇨ "null"
IFNULL("not null", "null")            ⇨ "not null"
```

- `INTERVAL(n,n1,n2,...)`

  Returnează 0 dacă `n < n1`, 1 dacă `n < n2` și așa mai departe, respectiv –1 dacă *n* este `NULL`. Valorile *n1*, *n2*, ... trebuie să fie în ordine strict crescătoare (`n1 < n2 < ...`), deoarece se folosește o căutare binară rapidă. În caz contrar, funcția `INTERVAL()` va prezenta un comportament imprevizibil.

```
INTERVAL(1.1,0,1,2)                   ⇨ 2
INTERVAL(7,1,3,5,7,9)                 ⇨ 4
```

- `ISNULL(expr)`

  Returnează 1 dacă valoarea expresiei `expr` este `NULL`; în caz contrar, returnează 0.

```
ISNULL(NULL)                          ⇨ 1
ISNULL(0)                             ⇨ 0
ISNULL(1)                             ⇨ 0
```

- `LEAST(expr1,expr2,...)`

  Returnează argumentul cel mai mic, unde noțiunea de „cel mai mic" este definită folosind aceleași reguli de comparație ca și cele pentru funcția `GREATEST()`.

```
LEAST(2,3,1)                          ⇨ 1
LEAST(38.5,94.2,-1)                   ⇨ -1.0
LEAST("a","ab","abc")                 ⇨ "a"
```

  Funcția `LEAST()` a fost introdusă în MySQL 3.22.5. În versiunile anterioare, puteți folosi în schimb funcția `MIN()`.

- `STRCMP(sir1,sir2)`

  Această funcție returnează 1, 0 sau –1 dacă primul argument este din punct de vedere lexicografic mai mare, egal, respectiv mai mic decât al doilea. Dacă unul dintre argumente este `NULL`, funcția returnează `NULL`.


```
STRCMP("a","a")                       ⇨ 0
STRCMP("a","A")                       ⇨ 1
STRCMP("A","a")                       ⇨ -1
```

Funcții numerice

Funcțiile numerice returnează `NULL` dacă se produce o eroare. De exemplu, dacă transferați funcției argumente care nu se află în interiorul domeniului de valori sau care sunt incorecte din alt motiv, funcția va returna `NULL`.

- `ABS(x)`

  Returnează valoarea absolută a lui *x*.

```
ABS(13.5)                             ⇨ 13.5
ABS(-13.5)                            ⇨ 13.5
```

- `ACOS(x)`

  Returnează valoarea arccos(*x*), respectiv `NULL` dacă *x* nu se află în domeniul cuprins între –1 și 1.

```
ACOS(1)                               ⇨ 0.000000
ACOS(0)                               ⇨ 1.570796
ACOS(-1)                              ⇨ 3.141593
```

  Funcția `ACOS()` a fost introdusă în versiunea MySQL 3.21.8.

- `ASIN(x)`

  Returnează valoarea arcsin(*x*), respectiv `NULL` dacă *x* nu se află în domeniul cuprins între –1 și 1.

```
ACOS(1)                               ⇨ 1.570796
ACOS(0)                               ⇨ 0.000000
ACOS(-1)                              ⇨ -1.570796
```

  Funcția `ASIN()` a fost introdusă în versiunea MySQL 3.21.8.

- `ATAN(x)`

  Returnează valoarea arctangentei lui *x*.

```
ATAN(1)                               ⇨ 0.785398
ATAN(0)                               ⇨ 0.000000
ATAN(-1)                              ⇨ -0.785398
```

  Funcția `ATAN()` a fost introdusă în MySQL 3.21.8.

- `ATAN2(x,y)`

  Este similară cu `ATAN(x,y)`, dar folosește semnele argumentelor pentru a determina cadranul valorii returnate (adică sfertul de cerc trigonometric în care se găsește aceasta – N.T.).

```
ATAN2(1,1)                            ⇨ 0.785398
ATAN2(1,-1)                           ⇨ 2.356194
ATAN2(-1,1)                           ⇨ -0.785398
ATAN2(-1,-1)                          ⇨ -2.356194
```

- `CEILING(x)`

  Returnează cel mai mic întreg care nu este mai mic decât *x*.

```
CEILING(3.8)          ⇨ 4
CEILING(-3.8)         ⇨ -3
```

- `COS(x)`

  Returnează cosinusul lui *x*, unde *x* este măsurat în radiani.

```
COS(0)                ⇨ 1.000000
COS((PI()/2))         ⇨ -1.000000
COS(PI()/2)           ⇨ 0.000000
```

  Funcția `COS()` a fost introdusă în versiunea MySQL 3.21.8.

- `COT(x)`

  Returnează cotangenta lui *x*, unde *x* este măsurat în radiani.

```
COT((PI()/2))         ⇨ 0.00000000
COT(PI()/4)           ⇨ 1.00000000
```

  Funcția `COT()` a fost introdusă în versiunea MySQL 3.21.16.

- `DEGREES(x)`

  Returnează valoarea lui *x*, convertită din radiani în grade.

```
DEGREES(PI())         ⇨ 180
DEGREES(PI()*2)       ⇨ 360
DEGREES(PI()/2)       ⇨ 90
DEGREES(-PI())        ⇨ -180
```

  Funcția `DEGREES` a fost introdusă în MySQL 3.21.16.

- `EXP(x)`

  Returnează valoarea e^x, unde *e* reprezintă baza logaritmilor naturali.

```
EXP(1)                ⇨ 2.718282
EXP(2)                ⇨ 7.389056
EXP(-1)               ⇨ 0.367879
1/EXP(1)              ⇨ 0.36787944
```

- `FLOOR(x)`

  Returnează cel mai mare întreg care nu este mai mare decât *x*.

```
FLOOR(3.8)            ⇨ 3
FLOOR(-3.8)           ⇨ -4
```

- `LOG(x)`

  Returnează logaritmul natural (în bază *e*) al lui *x*.

```
LOG(0)                ⇨ NULL
LOG(1)                ⇨ 0.000000
LOG(2)                ⇨ 0.693147
LOG(EXP(1))           ⇨ 1.000000
```

  Puteți folosi funcția `LOG()` pentru a calcula logaritmul lui *x* pentru o bază arbitrară *b*; folosiți expresia `LOG(x)/LOG(b)`.


```
LOG(100)/LOG(10)      ⇨ 2.00000000
LOG10(100)            ⇨ 2.000000
```

- `LOG10(x)`

  Returnează logaritmul lui *x* în baza 10.

```
LOG10(0)              ⇨ NULL
LOG10(10)             ⇨ 1.000000
LOG10(100)            ⇨ 2.000000
```

- `MOD(m,n)`

  Este identic cu `m % n`. Vezi „Operatori aritmetici".

- `PI()`

  Returnează valoarea lui π.

```
PI()                  ⇨ 3.141593
```

  Funcția `PI()` a fost introdusă în versiunea MySQL 3.21.8.

- `POW(x,y)`

  Returnează x^y, adică *x* ridicat la puterea *y*.

```
POW(2,3)              ⇨ 8.000000
POW(2,-3)             ⇨ 0.125000
POW(4,.5)             ⇨ 2.000000
POW(16,.25)           ⇨ 2.000000
```

- `POWER(x,y)`

  Această funcție este sinonimă cu funcția `POW()`. A fost introdusă în MySQL 3.21.16.

- `RADIANS(x,y)`

  Returnează valoarea lui *x*, convertită din grade în radiani.

```
RADIANS(0)            ⇨ 0
RADIANS(360)          ⇨ 6.28319
RADIANS(-360)         ⇨ -6.28319
```

  Funcția `RADIANS()` a fost introdusă în MySQL 3.21.16.

- `RAND()`

  `RAND(n)`

  `RAND()` returnează o valoare aleatoare cu virgulă mobilă, situată în intervalul cuprins între 0.0 și 1.0. Funcția `RAND(n)` execută aceeași operație, folosind *n* ca valoare de amorsă pentru utilitarul de selectare aleatoare. Toate apelurile cu aceeași valoare a lui *n* returnează același rezultat; puteți folosi această proprietate atunci când aveți nevoie de o secvență repetabilă de numere.

```
RAND(10)              ⇨ 0.181091
RAND(10)              ⇨ 0.181091
RAND()                ⇨ 0.117195
RAND()                ⇨ 0.358596
RAND(10)              ⇨ 0.181091
```

  În exemplele de mai sus, observați modul în care se comportă apelurile secvențiale la funcția `RAND()` atunci când funcția primește un argument, în comparație cu situațiile când acest argument nu este dat.

- ROUND(*x*)

  ROUND(*x*,*d*)

  Returnează valoarea lui *x*, rotunjită la un întreg. ROUND(*x*,*d*) returnează valoarea lui *x*, rotunjită la un număr cu d cifre după virgulă. Dacă d este 0, rezultatul nu are virgulă sau parte fracționară.

  ```
  ROUND(15.3)            ⇨ 15
  ROUND(15.5)            ⇨ 16
  ROUND(-33.27834,2)     ⇨ -33.28
  ROUND(1,4)             ⇨ 1.0000
  ```

- SIGN(*x*)

  Returnează –1, 0 sau 1, dacă valoarea lui *x* este negativă, zero sau pozitivă.

  ```
  SIGN(15.803)           ⇨ 1
  SIGN(0)                ⇨ 0
  SIGN(-99)              ⇨ -1
  ```

- SIN(*x*)

  Returnează valoarea sinusului lui *x*, unde *x* este măsurat în radiani.

  ```
  SIN(0)                 ⇨ 0.000000
  SIN(PI())              ⇨ 0.000000
  SIN(PI()/2)            ⇨ 1.000000
  ```

  Funcția SIN() a fost introdusă în versiunea MySQL 3.21.8.

- SQRT(*x*)

  Returnează rădăcina pătrată nenegativă a lui *x*.

  ```
  SQRT(625)              ⇨ 25.000000
  SQRT(2.25)             ⇨ 1.500000
  SQRT(-1)               ⇨ NULL
  ```

- TAN(*x*)

  Returnează valoarea tangentei lui x, unde x este măsurat în radiani.

  ```
  TAN(0)                 ⇨ 0.000000
  TAN((PI)/4)            ⇨ 1.000000
  ```

  Funcția TAN() a fost introdusă în versiunea MySQL 3.21.8.

- TRUNCATE(*x*,*d*)

  Returnează valoarea *x*, cu partea zecimală trunchiată la *d* cifre după virgulă. Dacă d este 0, rezultatul nu are virgulă sau parte zecimală. Dacă d este mai mare decât numărul de cifre după virgulă al argumentului *x*, partea zecimală este completată cu zerouri finale, până la dimensiunea dorită.

  ```
  TRUNCATE(1.23,1)       ⇨ 1.2
  TRUNCATE(1.23,0)       ⇨ 1
  TRUNCATE(1.23,4)       ⇨ 1.2300
  ```

  Funcția TRUNCATE() a fost introdusă în versiunea MySQL 3.21.16.

Funcții șir

Majoritatea funcțiilor din această secțiune returnează un rezultat sub formă de șir. Unele din acestea, precum LENGTH(), preiau ca argumente șiruri și returnează un număr. Pentru funcțiile care operează cu șiruri și sunt bazate pe pozițiile șirurilor, poziția primului caracter (de la stânga la dreapta) este 1, nu 0.

- ASCII(*sir*)

  Returnează codul ASCII al primului caracter al șirului sir. Returnează 0 dacă sir este vid, respectiv NULL dacă *sir* este NULL.

  ```
  ASCII("abc")           ⇨ 97
  ASCII("")              ⇨ 0
  ASCII(NULL)            ⇨ NULL
  ```

  Funcția ASCII() a fost introdusă în MySQL 3.21.2.

- BIN(*n*)

  Returnează valoarea lui *n* în formă binară, ca șir. Următoarele două expresii sunt echivalente:

  ```
  BIN(n)
  CONV(n,10,2)
  ```

  Pentru mai multe informații, consultați descrierea funcției CONV().

  Funcția BIN() a fost introdusă în versiunea MySQL 3.22.4.

- CHAR(*n1*,*n2*,...)

  Interpretează argumentele sub formă de coduri ASCII și returnează un șir alcătuit din concatenarea valorilor caracterelor corespunzătoare. Argumentele NULL sunt ignorate.

  ```
  CHAR(65)                     ⇨ "A"
  CHAR(97)                     ⇨ "a"
  CHAR(65,111,108,101,117,33)  ⇨ "Aoleu!"
  ```

  Funcția CHAR() a fost introdusă în versiunea MySQL 3.21.0.

- CHARACTER_LENGTH(*sir*)

  Această funcție este un sinonim al funcției LENGTH().

- CHAR_LENGTH(*sir*)

  Această funcție este un sinonim al funcției LENGTH().

- COALESCE(*expr1*,*expr2*,...)

  Returnează primul element diferit de NULL din listă.

  ```
  COALESCE(NULL,1/0,2,"a",45+97)   ⇨ "2"
  ```

  Această funcție a fost introdusă în MySQL versiunea 3.23.3.

- CONCAT(*sir1*,*sir2*,...)

  Returnează un șir alcătuit din concatenarea tuturor argumentelor sale. Returnează NULL dacă oricare dintre argumente este NULL. CONCAT() poate fi apelată cu mai multe argumente.

```
CONCAT("abc","def")               ⇨ "abcdef"
CONCAT("abc")                     ⇨ "abc"
CONCAT("abc", NULL)               ⇨ NULL
CONCAT("Salut",", ","pa")         ⇨ "Salut, pa"
```

- CONV(*n*, *din_baza*, *in_baza*)

  Dacă se dă un număr *n* reprezentat în baza *din_baza*, funcția returnează o reprezentare șir a numărului n în baza *in_baza*. Rezultatul este NULL dacă orice argument este NULL. *din_baza* și *in_baza* trebuie să fie numere întregi, cuprinse între 2 și 36. *n* este tratat ca o valoare BIGINT (întreg pe 64 de biți), dar poate fi specificat sub formă de șir, deoarece numerele din baze mai mari decât 10 pot conține cifre non-zecimale. (Acesta este și motivul pentru care funcția CONV() returnează un șir; rezultatul poate conține caractere de la A la Z pentru bazele 11-36.) Rezultatul este 0 dacă n nu este un număr corect exprimat în baza din_baza. (De exemplu, dacă *din_baza* este 16 și *n* este de forma "abcdefg", rezultatul este 0, deoarece g nu este o cifră hexazecimală corectă.)

  Caracterele non-zecimale din *n* pot fi specificate cu majuscule sau cu minuscule. Caracterele non-zecimale din rezultat vor fi scrise cu majuscule.

  n este tratat, în mod prestabilit, ca număr fără semn. Dacă specificați valoarea *in_baza* ca număr negativ, n este tratat ca număr cu semn.

  Conversia valorii 14, exprimată ca număr hexazecimal, în binar:

```
CONV("e",16,2)                    ⇨ "1110"
```

  Conversia în octal a valorii 255 exprimate în binar:

```
CONV(11111111,2,8)                ⇨ "377"
CONV("11111111",2,8)              ⇨ "377"
```

  Funcția CONV() a fost introdusă în versiunea MySQL 3.22.4.

- ELT(*n*,*sir1*,*sir2*,...)

  Returnează al *n*-lea șir din lista de șiruri *sir1*,*sir2*,.... Returnează NULL dacă nu există un șir la poziția respectivă sau dacă șirul respectiv sau *n* au valoarea NULL. Indexul primului șir este 1. Funcția ELT() este complementară cu funcția FIELD().

```
ELT(3,"a","b","c","d","e")        ⇨ "c"
ELT(0,"a","b","c","d","e")        ⇨ NULL
ELT(6,"a","b","c","d","e")        ⇨ NULL
ELT(FIELD("b","a","b","c"),"a","b","c")   ⇨ "b"
```

- EXPORT_SET(*n*,*on*,*off*,[*separator*,[*contor_biti*]])

  Returnează un șir alcătuit din șirurile on și off, separate prin șirul separator. on este folosit pentru fiecare bit cu valoarea 1 din n, iar off este folosit pentru fiecare bit cu valoarea 0. contor_biti indică numărul maxim de biți din n care trebuie examinați. Șirul separator prestabilit este o virgulă, iar valoarea prestabilită a argumentului contor_biti este 64.

```
EXPORT_SET(7,"+","-","",5)        ⇨ "+++--"
EXPORT_SET(0xa,"1","0","",6)      ⇨ "010100"
EXPORT_SET(97,"Y","N",",",8)      ⇨ "Y,N,N,N,N,Y,Y,N"
```

  Funcția EXPORT_SET() a fost introdusă în MySQL 3.23.2.

- FIELD(*sir*,*sir1*,*sir2*,...)

  Găsește șirul sir în lista de șiruri *str1*,*str2*... și returnează indexul șirului care corespunde. Returnează zero dacă nu există nici un șir echivalent sau dacă sir este NULL. Indexul primului șir este 1. FIELD() este complementar cu ELT().

```
FIELD("b","a","b","c")            ⇨ 2
FIELD("d","a","b","c")            ⇨ 0
FIELD(NULL,"a","b","c")           ⇨ 0
FIELD(ELT(2,"a","b","c"),"a","b","c")   ⇨ 2
```

- FIND_IN_SET(*sir*,*sir_lista*)

  sir_lista este un șir alcătuit din sub-șiruri separate prin virgule (adică similar unei valori SET). Funcția FIND_IN_SET() returnează indexul șirului *sir* din cadrul listei *sir_lista*. Returnează 0 dacă sir nu există în *sir_lista*, respectiv NULL dacă oricare dintre argumente este NULL. Indexul primului sub-șir este 1.

```
FIND_IN_SET("vaca","elan,vaca,porc")     ⇨ 2
FIND_IN_SET("caine","elan,vaca,porc")    ⇨ 0
```

  Funcția FIND_IN_SET() a fost introdusă în MySQL versiunea 3.21.22.

- FORMAT(X,D)

  Formatează numărul *x* la un șir cu d cifre zecimale, în formatul "nn,nnn.nnn"[1]. Dacă d este 0, rezultatul nu are nici un punct zecimal sau parte fracționară.

```
FORMAT(1234.56789,3)              ⇨ "1,234.568"
FORMAT(999999.99,2)               ⇨ "999,999.99"
FORMAT(999999.99,0)               ⇨ "1,000,000"
```

  Observați rotunjirea prezentată în ultimul exemplu din lista anterioară.

- HEX(*n*)

  Returnează valoarea lui *n* în format hexazecimal, sub formă de șir. Următoarele două expresii sunt echivalente:

```
HEX(n)
CONV(n,10,16)
```

  Pentru mai multe informații, consultați descrierea funcției CONV().

  Funcția HEX() a fost introdusă în MySQL 3.22.4.

- INSERT(*sir*,*poz*,*lung*,*sir_nou*)

  Returnează șirul *sir*, înlocuind sub-șirul care începe la poziția *poz* și are *lung* caractere lungime cu șirul *sir_nou*. Returnează șirul original dacă *poz* depășește domeniul, respectiv NULL dacă oricare dintre argumente este NULL.

```
INSERT("nighttime",6,4,"fall")    ⇨ "nightfall"
INSERT("sunshine",1,3,"rain or ") ⇨ "rain or shine"
INSERT("sunshine",0,3,"rain or ") ⇨ "sunshine"
```

- INSTR(*sir*,*subsir*)

  INSTR() este similar formei cu două argumente a funcției LOCATE(), dar cu argumentele inversate. Următoarele două expresii sunt echivalente:

---

[1] În limba engleză, virgula desparte ordinele de mărime, iar punctul are rol zecimal (invers decât în limba română). – N.T.

```
INSTR(sir,subsir)
LOCATE(subsir,sir)
```

- LCASE(*sir*)

Returnează șirul *sir* cu toate caracterele scrise cu minuscule, respectiv NULL dacă sir este NULL.

```
LCASE("New York, NY")        ⇨ "new york, ny"
LCASE(NULL)                  ⇨ NULL
```

- LEFT(*sir*,*lung*)

Returnează primele *lung* caractere (numărate de la stânga la dreapta) ale șirului *sir*, respectiv întregul șir, dacă acesta nu dispune de un număr atât de mare de caractere. Dacă *sir* este NULL, funcția returnează NULL. Dacă argumentul lung este NULL sau mai mic decât 1, funcția returnează un șir vid.

```
LEFT("piciorul meu stang", 2)   ⇨ "pi"
LEFT(NULL,10)                   ⇨ NULL
LEFT("abc",NULL)                ⇨ ""
LEFT("abc",0)                   ⇨ ""
```

- LENGTH(*sir*)

Returnează lungimea șirului *sir*.

```
LENGTH("abc")     ⇨ 3
LENGTH("")        ⇨ 0
LENGTH(NULL)      ⇨ NULL
```

- LOCATE(*subsir*,*sir*)

LOCATE(*subsir*,*sir*,*poz*)

Forma cu două argumente a funcției LOCATE() returnează poziția primei apariții a șirului *subsir* în cadrul șirului *sir*, respectiv 0 dacă *subsir* nu apare în cadrul șirului *sir*. Returnează NULL dacă orice argument este NULL. Dacă este dat argumentul de poziție *poz*, LOCATE() începe să caute *subsir* la poziția respectivă. Testul este sensibil la diferența între majuscule și minuscule.

```
LOCATE("b","abc")   ⇨ 2
LOCATE("b","ABC")   ⇨ 0
```

- LOWER(*sir*)

Această funcție este sinonimă cu LCASE().

- LPAD(*sir*,*lung*,*sir_comp*)

Returnează un șir alcătuit din valoarea șirului *sir*, completat la stânga cu șirul *sir_comp* pentru a se ajunge la un șir compus din *lung* caractere. Returnează *sir*, dacă acesta are deja *lung* caractere lungime.

```
LPAD("abc",12,"def")   ⇨ "defdefdefabc"
LPAD("abc",10,".")     ⇨ ".......abc"
LPAD("abc",2,".")      ⇨ "abc"
```

Funcția LPAD() a fost introdusă în versiunea MySQL 3.22.2.

- LTRIM(*sir*)

Returnează șirul *sir* fără spațiile de început, respectiv NULL dacă sir este NULL:


```
LTRIM("  abc  ")   ⇨ "abc  "
```

- MAKE_SET(*n*,*bit0_sir*,*bit1_sir*,...)

Construiește o valoare SET (un șir alcătuit din sub-șiruri separate prin virgule) bazată pe valoarea argumentului întreg *n* și pe șirurile *bit0_sir*, *bit1_sir*, ... Pentru fiecare bit activat din valoarea lui *n*, șirul corespunzător este inclus în rezultat. (Dacă bitul 0 este activat, rezultatul include *bit0_sir* etc.) Dacă *n* este 0, rezultatul este șirul vid. Dacă *n* este NULL, rezultatul este NULL. Dacă orice șir din listă este NULL, acesta este ignorat la construirea șirului rezultat:

```
MAKE_SET(8,"a","b","c","d","e")      ⇨ "d"
MAKE_SET(7,"a","b","c","d","e")      ⇨ "a,b,c"
MAKE_SET(2+16,"a","b","c","d","e")   ⇨ "b,e"
MAKE_SET(2|16,"a","b","c","d","e")   ⇨ "b,e"
MAKE_SET(-1,"a","b","c","d","e")     ⇨ "a,b,c,d,e"
```

Ultimul exemplu selectează toate șirurile, deoarece toți biții valorii –1 sunt activați.

Funcția MAKE_SET() a fost introdusă în versiunea MySQL 3.22.2.

- MID(*sir*,*poz*,*lung*)

Returnează un sub-șir al șirului *sir*, care începe la poziția *poz* și are lungimea de *lung* caractere. Dacă oricare dintre argumente este NULL, returnează NULL.

```
MID("ce exemplu plicticos",12,9)   ⇨ "plicticos"
```

- OCT(*n*)

Returnează valoarea lui *n* în formă octală, ca șir. Următoarele două expresii sunt echivalente:

```
OCT(n)
CONV(n,10,8)
```

Pentru mai multe informații, consultați descrierea funcției CONV().

Funcția OCT() a fost introdusă în versiunea MySQL 3.22.4.

- OCTET_LENGTH(*sir*)

Această funcție este un sinonim pentru LENGTH().

- POSITION(*subsir* IN *sir*) POSITION()

Această funcție este similară formei cu două argumente a funcției LOCATE(). Următoarele expresii sunt echivalente:

```
POSITION(subsir IN sir)
LOCATE(subsir,sir)
```

- REPEAT(*sir*,*n*)

Returnează un șir alcătuit din șirul *sir* repetat de *n* ori. Returnează un șir vid dacă *n* nu este pozitiv, respectiv NULL dacă oricare dintre argumente este NULL.

```
REPEAT("x",10)      ⇨ "xxxxxxxxxx"
REPEAT("abc",3)     ⇨ "abcabcabc"
```

- REPLACE(*sir*,*din_sir*,*in_sir*)

Returnează un șir alcătuit din șirul *sir*, unde toate aparițiile șirului *din_sir* înlocuite cu șirul *in_sir*. Returnează NULL dacă oricare dintre argumente este NULL. Dacă *to_sir* este vid, efectul constă în ștergerea tuturor aparițiilor șirului *din_sir*. Dacă *din_sir* este vid, funcția REPLACE() returnează șirul *sir* nemodificat.

```
REPLACE("abracadabra","a","oh")      ⇨ "ohbrohcohdohbroh"
REPLACE("abracadabra","a","")        ⇨ "brcdbr"
REPLACE("abracadabra","","oh")       ⇨ "abracadabra"
```

- REVERSE(*sir*)

  Returnează un şir alcătuit din şirul sir cu caracterele scrise în ordine inversă. Returnează NULL dacă şirul sir este NULL.

```
REVERSE("abracadabra")       ⇨ "arbadacarba"
REVERSE("Madam, I'm Adam")   ⇨ "madA m'I ,madaM"
```

  Funcţia REVERSE() a fost introdusă în versiunea MySQL 3.21.19.

- RIGHT(*sir*,*lung*)

  Returnează primele *lung* caractere ale şirului *sir*, numărate de la dreapta la stânga, respectiv întregul şir, dacă acesta nu conţine un număr atât de mare de caractere. Returnează NULL dacă şirul este NULL. Returnează şirul vid dacă *lung* este NULL sau dacă este mai mic decât 1.

```
RIGHT("parasuta",4)   ⇨ "suta"
```

- RPAD(*sir*,*lung*,*sir_compl*)

  Returnează un şir alcătuit din valoarea şirului *sir*, completat la dreapta cu şirul *sir_comp* pentru a ajunge la un şir compus din *lung* caractere. Returnează *sir*, dacă acesta are deja *lung* caractere lungime.

```
LPAD("abc",12,"def")   ⇨ "abcdefdefdef"
LPAD("abc",10,".")     ⇨ "abc......."
LPAD("abc",2,".")      ⇨ "abc"
```

  Funcţia LPAD() a fost introdusă în versiunea MySQL 3.22.2.

- RTRIM(*sir*)

  Returnează şirul *sir* fără spaţiile de sfârşit, respectiv NULL dacă *sir* este NULL.

```
LTRIM("    abc    ")   ⇨ "    abc"
```

- SOUNDEX(*sir*)

  Returnează un şir codificat cu algoritmul Soundex[2], calculat pornind de la şirul *sir*. Caracterele non-alfanumerice din *sir* sunt ignorate. Caracterele internaţionale non-alfabetice din afara domeniului cuprins între A şi Z sunt tratate ca vocale.

```
SOUNDEX("Cow")     ⇨ "C000"
SOUNDEX("Cowl")    ⇨ "C400"
SOUNDEX("Howl")    ⇨ "H400"
SOUNDEX("Hello")   ⇨ "H400"
```

- SPACE(*n*)

  Returnează un şir alcătuit din *n* spaţii, respectiv şirul vid dacă *n* este non-pozitiv sau NULL dacă *n* este NULL.

```
SPACE(6)      ⇨ "      "
```

[2] Pentru semnificaţia funcţiei Soundex şi modul de utilizare a acesteia, vezi capitolul 13 al volumului *Visual Basic 6 – Baze de date*, apărut recent la Editura Teora. – N.T.


```
SPACE(0)      ⇨ ""
SPACE(NULL)   ⇨ NULL
```

  Funcţia SPACE() a fost introdusă în MySQL 3.21.16.

- SUBSTRING(*sir*,*poz*)

  SUBSTRING(*sir*,*poz*,*lung*)

  SUBSTRING(*sir* FROM *poz*)

  SUBSTRING(*sir* FROM *poz* FOR *lung*)

  Returnează un sub-şir al şirului *sir*, începând de la poziţia *poz*. Dacă este dat un argument *lung*, returnează un sub-şir care are un număr de caractere egal cu lung; în caz contrar, returnează întreaga parte dreaptă a şirului *sir*, începând de la poziţia poz.

```
SUBSTRING("abcdef",3)     ⇨ "cdef"
SUBSTRING("abcdef",3,2)   ⇨ "cd"
```

  Următoarele expresii sunt echivalente:

```
SUBSTRING(sir,poz,lung)
SUBSTRING(sir FROM poz FOR lung)
MID(sir,poz,lung)
```

- SUBSTRING_INDEX(*sir*,*delim*,*n*)

  Returnează un sub-şir al şirului *sir*. Dacă *n* este pozitiv, SUBSTRING_INDEX() găseşte cea de-a n-a apariţie a şirului de delimitare *delim*, apoi returnează toate caracterele aflate la stânga acelui delimitator. Dacă n este negativ, SUBSTRING_INDEX găseşte a n-a apariţie a şirului *delim*, numărând de la capătul din dreapta al şirului *sir*, apoi returnează toate caracterele din dreapta delimitatorului respectiv. Dacă *delim* nu se află în *sir*, atunci va fi returnat întregul şir. Returnează NULL dacă orice argument are valoarea NULL.

```
SUBSTRING_INDEX("jar-jar","j",-2)   ⇨ "ar-jar"
SUBSTRING_INDEX(USER(),"@",1)       ⇨ "paul"
```

  Funcţia SUBSTRING_INDEX() a fost introdus în versiunea MySQL 3.21.15.

- TRIM([[LEADING | TRAILING | BOTH] [*trim_sir*] FROM] *sir*)

  Returnează şirul sir fără instanţele iniţiale şi/sau finale ale şirului *trim_sir*. Dacă este specificată opţiunea LEADING, funcţia TRIM() elimină apariţiile iniţiale ale şirului *trim_sir*. Dacă este specificată opţiunea TRAILING, funcţia TRIM() elimină apariţiile finale ale şirului *trim_sir*. Dacă este specificată opţiunea BOTH, funcţia TRIM() elimină apariţiile iniţiale şi finale ale şirului *trim_sir*. Opţiunea prestabilită este BOTH dacă nu este specificată nici una dintre opţiunile LEADING, TRAILING sau BOTH. Dacă nu este specificat *trim_sir*, funcţia TRIM() elimină spaţiile.

```
TRIM("^" FROM "^^^XYZ^^")            ⇨ "XYZ"
TRIM(LEADING "^" FROM "^^^XYZ^^")    ⇨ "XYZ^^"
TRIM(TRAILING "^" FROM "^^^XYZ^^")   ⇨ "^^^XYZ"
TRIM(BOTH "^" FROM "^^^XYZ^^")       ⇨ "XYZ"
TRIM(BOTH FROM "    ABC    ")        ⇨ "ABC"
TRIM("    ABC    ")                  ⇨ "ABC"
```

  Funcţia TRIM() a fost introdusă în MySQL 3.21.12.

- UCASE(*sir*)

  Returnează șirul *sir* cu toate caracterele convertite la majuscule, respectiv NULL dacă *sir* este NULL.

```
UCASE("New York, NY")          ⇨ "NEW YORK, NY"
UCASE(NULL)                    ⇨ NULL
```

- UPPER(sir)

  Această funcție este sinonimă cu UCASE().

Funcții de tip dată și oră

Funcțiile de tip dată și oră preiau diferite tipuri de argumente. În general, o funcție care presupune un argument DATE va accepta, de asemenea, un argument DATETIME sau TIMESTAMP și va ignora partea de oră a valorii.

- ADDATE(*data*,INTERVAL *expr interval*)

  Această funcție este sinonimă cu DATE_INTERVAL().

- CURDATE()

  Returnează data curentă sub formă de șir în formatul "AAAA-LL-ZZ", respectiv ca un număr AAAALLZZ, conform contextului în care este folosită funcția:

```
CURDATE()                      ⇨ "1999-08-10"
CURDATE() + 0                  ⇨ 19990810
```

Valori valabile pentru dată și oră

Dacă nu furnizați valori valabile pentru dată și oră funcțiilor care acceptă atare argumente, nu vă așteptați la un rezultat rezonabil. Mai întâi verificați argumentele.

- CURRENT_DATE

  Această funcție este un sinonim pentru CURDATE(); observați lipsa parantezelor.

- CURRENT_TIME

  Această funcție este un sinonim pentru CURTIME(); observați lipsa parantezelor.

- CURRENT_TIMESTAMP

  Această funcție este sinonimă cu NOW(); observați lipsa parantezelor.

- CURTIME()

  Returnează ora curentă din zi sub formă de șir în formatul "hh:mm:ss", respectiv sub formă de număr în formatul hhmmss, conform contextului în care este folosită funcția.

```
CURTIME()                      ⇨ "16:41:01"
CURTIME() + 0                  ⇨ 164101
```

  Funcția CURTIME() a fost introdusă în MySQL 3.21.12.

- DATE_ADD(*data*,INTERVAL *expr interval*)

  Preia o valoare de tip dată, respectiv dată-oră data, adaugă la aceasta un interval de timp și returnează rezultatul. expr specifică valoarea orei care trebuie adăugată la data (respectiv scăzută, dacă expr începe cu semnul minus), iar interval specifică modul

de interpretare a intervalului. Rezultatul este o valoare DATE dacă data este o valoare de tip DATE și dacă în calculul rezultatului nu sunt implicate valori referitoare la oră. Dacă această afirmație nu este adevărată, rezultatul este o valoare DATETIME. Rezultatul este NULL dacă data nu este o dată corectă.

```
DATE_ADD("1999-12-01",INTERVAL 1 YEAR)              ⇨ "2000-12-01"
DATE_ADD("1999-12-01",INTERVAL 60 DAY)              ⇨ "2000-01-30"
DATE_ADD("1999-12-01",INTERVAL -3 MONTH)            ⇨ "1999-09-01"
DATE_ADD("1999-12-01 08:30:00",INTERVAL 12 HOUR)    ⇨ "1999-09-01
                                                       20:30:00"
```

Tabelul C.2 prezintă valorile permise pentru argumentul interval, semnificațiile lor și formatul în care trebuie specificate valorile pentru fiecare tip de interval. Cuvântul cheie INTERVAL și specificatorii interval pot fi menționați folosind atât majuscule, cât și minuscule.

Expresia expr care se adaugă la dată poate fi specificată sub formă de număr sau ca un șir, cu excepția situațiilor în care conține caractere altele decât cifrele, caz în care expresia trebuie specificată sub formă de șir. Caracterele de delimitare sunt arbitrare și pot fi alcătuite din orice semn de punctuație:

```
DATE_ADD("1999-12-01",INTERVAL "2:3" YEAR_MONTH) ⇨ "2002-03-01"
DATE_ADD("1999-12-01",INTERVAL "2-3" YEAR_MONTH) ⇨ "2002-03-01"
```

Componentele valorii expr sunt comparate de la dreapta la stânga cu părțile așteptate, în funcție de specificatorul interval. De exemplu, formatul așteptat pentru HOUR_SECOND este "hh:mm:ss". O valoare expr de "15:21" este interpretată sub forma "00:15:21", nu "15:21:00".

```
DATE_ADD("1999-12-01 12:00:00",INTERVAL "15:21" HOUR_SECOND)
                                         ⇨ "1999-12-01 12:15:21"
```

Tabelul C.2 Tipurile de interval pentru funcția DATE_ADD()

Tip	Semnificație	Formatul valorii
SECOND	Secunde	ss
MINUTE	Minute	mm
HOUR	Ore	hh
DAY	Zile	ZZ
MONTH	Luni	LL
YEAR	Ani	AA
MINUTE_SECOND	Minute și secunde	"mm:ss"
HOUR_MINUTE	Ore și minute	"hh:mm"
HOUR_SECOND	Ore, minute și secunde	"hh:mm:ss"
DAY_HOUR	Zile și ore	"ZZ hh"
DAY_MINUTE	Zile, ore și minute	"ZZ hh:mm"
DAY_SECOND	Zile, ore, minute și secunde	"ZZ hh:mm:ss"
YEAR_MONTH	Ani și luni	"AA-LL"

Dacă *interval* este YEAR, MONTH sau YEAR_MONTH, iar partea rezultatului corespunzătoare zilei este mai mare decât numărul de zile din luna rezultantă, ziua primește valoarea numărului maxim de zile din luna respectivă.

```
DATE_ADD("1999-12-31",INTERVAL 2 MONTH)    ⇨ "2000-02-29"
```

Funcția DATE_ADD() a fost introdusă în versiunea MySQL 3.22.4.

- DATE_FORMAT(*data*, *format*)

Formatează o valoare data de tip dată sau dată-oră în conformitate cu șirul de formatare format și returnează șirul rezultant. DATE_FORMAT() se poate folosi pentru a reformata valorile DATE sau DATETIME din forma folosită de MySQL cu scopul de a furniza orice format doriți.

```
DATE_FORMAT("1999-12-01","%M %e, %Y")       ⇨ "December 1, 1999"
DATE_FORMAT ("1999-12-01","The %D of %M") ⇨ "The 1st of December"
```

Tabelul C.3 prezintă specificatorii disponibili care se pot folosi în șirul de formatare.

Începând de la versiunea MySQL 3.23.0, este necesară utilizarea caracterului % anterior codului de formatare. În versiunile anterioare, acest caracter este permis, dar are un caracter opțional. Caracterele prezente în șirul de formatare care nu sunt prezente în tabel sunt copiate literal în șirul rezultat.

Dacă faceți referire la specificatori de oră pentru o valoare DATE, partea de oră a valorii este tratată ca "00:00:00".

```
DATE_FORMAT("1999-12-01","%i")              ⇨ "00"
```

Funcția DATE_FORMAT() a fost introdusă în versiunea MySQL 3.21.14.

Tabelul C.3 Specificatori de formatare pentru funcția DATE_FORMAT()

Specificator	Semnificație
%S, %s	Secundă în format numeric cu două cifre (00, 01, ..., 59)
%i	Minut în format numeric cu două cifre (00, 01, ..., 59)
%H	Oră în format numeric cu două cifre, 24 de ore (00, 01, ..., 23)
%h, %I	Oră în format numeric cu două cifre, 12 ore (01, 02, ..., 12)
%k	Oră în format numeric, 24 de ore (0, 1,... , 23)
%l	Oră în format numeric, 12 ore (1, 2,... , 12)
%T	Oră din zi în format 24 de ore (*hh:mm:ss*)
%r	Oră din zi, în format 12 ore (*hh:mm:ss* AM sau *hh:mm:ss* PM)
%p	AM sau PM
%W	Numele zilei din săptămână (Sunday, Monday, ..., Saturday)
%a	Numele abreviat al zilei din săptămână (Sun, Mon, ..., Sat[3])
%d	Ziua din lună, în format numeric cu două cifre (00, 01, ..., 31)
%e	Ziua din lună, în format numeric (1, 2, ..., 31)
%D	Ziua din lună cu sufixele limbii engleze (1st, 2nd, 3rd, ...)
%w	Ziua din săptămână, în format numeric (0=duminică, 1=luni, ..., 6=sâmbătă)
%j	Ziua din an, în format numeric cu trei cifre (001, 002, ..., 366)
%U	Săptămâna (0, 1..., 52), unde duminica este prima zi a săptămânii
%u	Săptămâna (0, 1, ..., 52), unde lunea este prima zi a săptămânii
%M	Numele lunii (January, February, ..., December)
%b	Numele lunii, în formă abreviată (Jan, Feb, ..., Dec)
%m	Luna în format numeric cu două cifre (01, 02, ..., 12)
%c	Luna în format numeric (1, 2, ..., 12)
%Y	Anul, în format numeric cu patru cifre
%y	Anul, în format numeric cu două cifre
%%	Un caracter procent literal

[3] Numele zilelor din săptămână, respectiv ale lunilor din an, sunt returnate de program în limba engleză. – N.T.

- DATE_SUB(*data*,INTERVAL *expr interval*)

Execută operații aritmetice cu date în aceeași manieră ca și DATE_ADD(), cu excepția faptului că expr se scade din valoarea datei data. Pentru mai multe informații, consultați informațiile despre funcția DATE_ADD().

```
DATE_SUB("1999-12-01",INTERVAL 1 MONTH)     ⇨ "1999-11-01"
DATE_SUB("1999-12-01","INTERVAL "13-2" YEAR_MONTH)
                                            ⇨ "1986-10-01"
DATE_SUB("1999-12-01 04:53:12",INTERVAL "13-2" MINUTE_SECOND)
                                            ⇨ "1999-12-01 04:40:10"
DATE_SUB("1999-12-01 04:53:12",INTERVAL "13-2" HOUR_MINUTE)
                                            ⇨ "1999-12-01 15:51:12"
```

Funcția DATE_SUB() a fost introdusă în MySQL versiunea 3.22.4.

- DAYNAME(*data*)

Returnează un șir care conține numele zilei din săptămână (în limba engleză – N.T.) pentru valoarea de tip dată *data*.

```
DAYNAME("1999-12-01")                       ⇨ "Wednesday"
DAYNAME("1900-12-01")                       ⇨ "Saturday"
```

Funcția DAYNAME() a fost introdusă în MySQL versiunea 3.21.23.

- DAYOFMONTH(*data*)

Returnează valoarea numerică a zilei din lună corespunzătoare valorii de tip dată *data*, în domeniul cuprins între 1 și 31.

```
DAYOFMONTH("1999-12-01")                    ⇨ "1"
DAYOFMONTH("1999-12-25")                    ⇨ "25"
```

Funcția DAYOFMONTH() a fost introdusă în MySQL versiunea 3.21.22.

- DAYOFWEEK(*data*)

  Returnează valoarea numerică a zilei din săptămână corespunzătoare valorii de tip dată *data*. Valorile zilelor din săptămână variază între 1 (duminică) și 7 (sâmbătă), conform standardului ODBC. Vezi și funcția WEEKDAY().

```
DAYOFWEEK("1999-12-05")          ⇨ 1
DAYNAME("1999-12-05")            ⇨ "Sunday"
DAYOFWEEK("1999-12-11")          ⇨ 7
DAYNAME("1999-12-05")            ⇨ "Saturday"
```

  Funcția DAYOFWEEK() a fost introdusă în MySQL versiunea 3.21.15.

- DAYOFYEAR(*data*)

  Returnează valoarea numerică a zilei din an aferente valorii de tip dată *data*, unde valoarea zilei din an este cuprinsă între 1 și 366.

```
DAYOFYEAR("1999-12-01")          ⇨ 335
DAYOFYEAR("2000-12-31")          ⇨ 366
```

- EXTRACT(*interval* FROM *dataora*)

  Returnează partea din valoarea dată-oră *dataora* indicată de *interval*, care poate fi oricare dintre specificatorii de interval permiși pentru DATE_ADD().

```
EXTRACT(YEAR FROM "1999-12-01 13:42:19")          ⇨ 1999
EXTRACT(MONTH FROM "1999-12-01 13:42:19")         ⇨ 12
EXTRACT(DAY FROM "1999-12-01 13:42:19")           ⇨ 1
EXTRACT(HOUR_MINUTE FROM "1999-12-01 13:42:19")   ⇨ 1342
EXTRACT(SECOND FROM "1999-12-01 13:42:19")        ⇨ 19
```

  EXTRACT() poate fi de asemenea folosit cu valorile DATE; în acest caz, extragerea unei ore va determina rezultatul NULL.

  Funcția EXTRACT() a fost introdusă începând cu MySQL versiunea 3.23.0.

- FROM_DAYS(*n*)

  Dacă este dată o valoare numerică reprezentând numărul de zile scurse de la anul 0 (care se obțin, în mod caracteristic, apelând funcția TO_DAYS()), returnează data corespunzătoare.

```
TO_DAYS("1999-12-01")            ⇨ 730454
FROM_DAYS(730454 + 3)            ⇨ "1999-12-04"
```

  Funcția FROM_DAYS() este destinată numai pentru datele care se conformează calendarului gregorian (începând din anul 1582).

- FROM_UNIXTIME(*unix_timestamp*)

  FROM_UNIXTIME(*unix_timestamp*, *format*)

  Dacă este dată o valoare UNIX de tip amprentă de timp *unix_timestamp*, cum este cea returnată de funcția UNIX_TIMESTAMP(), funcția returnează o valoare de tip dată-oră în formatul "AAAA-LL-ZZ hh:mm:ss" sau un număr în format AAAALLZZhhmmss, conform contextului în care se folosește funcția. Dacă este dat argumentul *format*, valoarea returnată este formatată sub formă de șir, așa cum se procedează în cazul funcției DATE_FORMAT().


```
FROM_UNIXTIME(934340541)         ⇨ "1999-08-10 22:02:21"
FROM_UNIXTIME(944028000)         ⇨ "1999-12-01 00:00:00"
FROM_UNIXTIME(944028000, "%Y")   ⇨ "1999"
```

  Funcția FROM_UNIXTIME() a fost introdusă în MySQL versiunea 3.21.5. Forma cu două argumente a fost introdusă în MySQL versiunea 3.21.8.

- HOUR(*ora*)

  Returnează valoarea numerică a orei pentru valoarea de tip oră *ora*, valoare cuprinsă în intervalul 0-23.

```
HOUR("12:31:58")                 ⇨ 12
```

  Funcția HOUR() a fost introdusă în MySQL versiunea 3.21.22.

- MINUTE(*ora*)

  Returnează valoarea numerică a minutelor pentru valoarea de tip oră *ora*, valoare cuprinsă în intervalul 0-59.

```
MINUTE("12:31:58")               ⇨ 31
```

  Funcția MINUTE() a fost introdusă în MySQL versiunea 3.21.22.

- MONTH(*data*)

  Returnează valoarea numerică a lunii din an aferente valorii de tip dată *data*, valoare cuprinsă în intervalul 1-12.

```
MONTH("1999-12-01")              ⇨ 12
```

  Funcția MONTH() a fost introdusă în MySQL versiunea 3.21.22.

- MONTHNAME(*data*)

  Returnează un șir conținând numele lunii (în limba engleză – N.T.) aferente valorii de tip dată *data*.

```
MONTHNAME("1999-12-01")          ⇨ "December"
```

  Funcția MONTHNAME() a fost introdusă în MySQL versiunea 3.21.23.

- NOW()

  Returnează data și ora curentă sub formă de șir, în formatul "AAAA-LL-ZZ hh:mm:ss", respectiv sub formă de număr în formatul AAAALLZZhhmmss, conform contextului în care este folosită funcția.

```
NOW()                            ⇨ "1999-08-10 18:51:43"
NOW() + 0                        ⇨ 19990810185143
```

- PERIOD_ADD(*perioada*, *n*)

  Adaugă n luni la valoarea de tip perioadă *perioada* și returnează rezultatul. Formatul valorii returnate este AAAALL. Formatul argumentului perioada poate fi AAAALL sau AALL. Nici una nu este o valoare de tip dată.

```
PERIOD_ADD(199902,12)            ⇨ 200002
PERIOD_ADD(9902,-3)              ⇨ 199811
```

- PERIOD_DIFF(*perioada1*, *perioada2*)

  Preia diferența între argumentele de tip perioadă și returnează numărul de luni cuprins între acestea. Argumentele pot avea formatul AAAALL sau AALL. Nici unul nu este o valoare de tip dată.

```
PERIOD_DIFF(200002,199902)        ⇨ 12
PERIOD_DIFF(199811,9902)          ⇨ -3
```

- `QUARTER(data)`

Returnează valoarea numerică a trimestrului din an aferent valorii de tip dată *data*, în domeniul cuprins între 1 și 4.

```
QUARTER("1999-12-01")             ⇨ 4
QUARTER("2001-01-01")             ⇨ 1
```

Funcția `QUARTER()`a fost introdusă începând de la MySQL versiunea 3.21.22.

- `SECOND(ora)`

Returnează valoarea numerică a numărului de secunde pentru valoarea de tip oră *ora*, în domeniul cuprins între 0-59.

```
SECOND("12:31:58")                ⇨ 58
SECOND(123158)                    ⇨ 58
```

Funcția `SECOND()` a fost introdusă în versiunea MySQL 3.21.22.

- `SEC_TO_TIME(secunde)`

Dacă este dat un număr de secunde *secunde*, returnează valoarea corespondentă a orei sub formă de șir în formatul `"hh:mm:ss"`, respectiv sub formă de număr în formatul `hhmmss`, conform contextului în care este folosită funcția.

```
SEC_TO_TIME(29834)                ⇨ "08:17:14"
SEC_TO_TIME(29834) + 0            ⇨ 81714
```

Funcția `SEC_TO_TIME()` a fost introdusă în versiunea MySQL 3.21.5.

- `SUBDATE(data,INTERVAL expr interval)`

Această funcție este un sinonim pentru `DATE_SUB()`.

- `SYSDATE()`

Această funcție este un sinonim pentru `NOW()`.

- `TIME_FORMAT(ora,format)`

Formatează valoarea de tip oră *ora* în conformitate cu șirul de formatare *format* și returnează șirul rezultant. Șirul de formatare este similar celui folosit de către funcția `DATE_FORMAT()`, dar singurii specificatori care pot fi folosiți sunt cei referitori la oră. Utilizarea altor specificatori determină o valoare `NULL` sau 0.

```
TIME_FORMAT("12:31:58","%H %i")   ⇨ "12 31"
TIME_FORMAT(123158,"%H %i")       ⇨ "12 31"
```

Funcția `TIME_FORMAT()` a fost introdusă în MySQL versiunea 3.21.3.

- `TIME_TO_SEC(ora)`

Dacă este dată o valoare *ora* care reprezintă timpul scurs, returnează un număr care reprezintă numărul corespunzător de secunde.

```
TIME_TO_SEC("08:17:14")           ⇨ 29834
```

Funcția `TIME_TO_SEC()` a fost introdusă în MySQL versiunea 3.21.16.

- `TO_DAYS(data)`

Returnează o valoare numerică reprezentând valoarea de tip dată *data* convertită la numărul de zile scurs de la anul 0. Valoarea poate fi transferată funcției `FROM_DAYS()` pentru a o reconverti înapoi într-o dată.

```
TO_DAYS("1999-12-01")             ⇨ 730454
FROM_DAYS(730454 - 365)           ⇨ "1999-12-01"
```

Funcția `TO_DAYS()` este destinată numai datelor acoperite de calendarul gregorian (de la anul 1582).

- `UNIX_TIMESTAMP()`

`UNIX_TIMESTAMP(data)`

Când este apelată fără argumente, returnează numărul de secunde de la începutul epocii UNIX (`"1970-01-01 00:00:00"` GMT). Când este apelată cu un argument de tip dată *data*, returnează numărul de secunde între epocă și data respectivă. *data* poate fi specificat în mai multe moduri: sub formă de șir `DATE` sau `DATETIME`, sub formă de valoare `TIMESTAMP` sau sub formă de număr în formatul `AAAALLZZ` sau `AALLZZ`, ora locală.

```
UNIX_TIMESTAMP()                  ⇨ 934341073
UNIX_TIMESTAMP("1999-12-01")      ⇨ 944028000
UNIX_TIMESTAMP(991201)            ⇨ 944028000
```

- `WEEK(data)`

`WEEK(data,prima)`

Când este apelată cu un singur argument, returnează un număr care reprezintă săptămâna din an aferentă valorii de tip dată *data*, în domeniul cuprins între 0 și 52. Se presupune că duminica este prima zi a săptămânii. Când este apelat cu două argumente, returnează același tip de valoare, dar argumentul *prima* indică prima zi din săptămână. Dacă *prima* este 0, săptămâna începe duminică. Dacă *prima* este 1, săptămâna începe luni.

```
WEEK("1999-12-05")                ⇨ 49
WEEK("1999-12-05",0)              ⇨ 49
WEEK("1999-12-05",1)              ⇨ 48
```

Funcția `WEEK()` a fost introdusă în versiunea MySQL 3.21.22. Forma cu două argumente a fost introdusă în MySQL versiunea 3.22.1.

- `WEEKDAY(data)`

Returnează valoarea numerică a zilei din săptămână aferentă valorii de tip dată `data`. Valorile zilelor din săptămână sunt cuprinse între 0 (pentru luni) și 6 (pentru duminică); vezi și funcția `DAYOFWEEK()`.

```
WEEKDAY("1999-12-05")             ⇨ 6
DAYNAME("1999-12-05")             ⇨ "Sunday"
WEEKDAY("1999-12-13")             ⇨ 0
DAYNAME("1999-12-05")             ⇨ "Monday"
```

- `YEAR(data)`

Returnează valoarea numerică a anului pentru valoarea de tip dată `data`, în domeniul cuprins între 1000 și 9999.

```
YEAR("1999-12-01")                ⇨ 1999
```

Funcția `YEAR()` a fost introdusă în versiunea MySQL 3.21.22.

Funcții de sumar

Funcțiile de sumar sunt de asemenea cunoscute sub numele de *funcții agregat*. Acestea calculează o valoare unică, pe baza unui grup de valori. Cu toate acestea, valoarea rezultantă este bazată numai pe valori diferite de NULL din rândurile selectate (cu excepția faptului că funcția COUNT(*) numără toate rândurile). Funcțiile de sumar se pot folosi pentru a rezuma coloane întregi, respectiv valori atunci când sunt grupate în funcție de valori distincte într-o altă coloană sau combinație de coloane. Vezi secțiunea „Generarea sumarelor" în capitolul 1, „Introducere în MySQL și SQL".

Pentru exemplele din această secțiune, vom presupune că aveți un tabel numit tabelul_meu, cu o coloană de valori întregi denumită coloana_mea, care conține șase rânduri, cu valorile 1, 3, 5, 7, 9 și NULL.

- AVG(*expr*)

  Returnează valoarea medie a argumentului *expr* pentru toate valorile diferite de NULL din rândurile selectate.

  ```
  SELECT AVG(coloana_mea) FROM tabelul_meu       ⇨ 5.0000
  SELECT AVG(coloana_mea)*2 FROM tabelul_meu     ⇨ 10.0000
  SELECT AVG(coloana_mea*2) FROM tabelul_meu     ⇨ 10.0000
  ```

- BIT_AND(*expr*)

  Returnează valoarea ȘI la nivel de bit a argumentului *expr* pentru toate valorile diferite de NULL din rândurile selectate.

  ```
  SELECT BIT_AND(coloana_mea) FROM tabelul_meu   ⇨ 1
  ```

  Funcția BIT_AND() a fost introdusă în MySQL versiunea 3.21.11.

- BIT_OR(*expr*)

  Returnează valoarea SAU la nivel de bit a argumentului *expr* pentru toate valorile diferite de NULL din rândurile selectate.

  ```
  SELECT BIT_OR(coloana_mea) FROM tabelul_meu    ⇨ 15
  ```

  Funcția BIT_OR() a fost introdusă în MySQL versiunea 3.21.11.

- COUNT(*expr*)

  Cu orice argument diferit de *, returnează un număr al valorilor diferite de NULL din setul de rezultate. Cu argumentul *, returnează numărul tuturor rândurilor din setul de rezultate, indiferent dacă rândurile conțin sau nu valori NULL.

  ```
  SELECT COUNT(coloana_mea) FROM tabelul_meu     ⇨ 5
  SELECT COUNT(*) FROM tabelul_meu               ⇨ 5
  ```

  Funcția COUNT(*) fără clauza WHERE este optimizată astfel încât să returneze foarte rapid numărul de înregistrări din tabelul denumit în clauza FROM. Când sunt specificate mai multe tabele, COUNT(*) returnează produsul numerelor de rânduri din tabelele individuale:

  ```
  SELECT COUNT(*) FROM tabelul_meu AS m1, tabelul_meu AS m2
                                                 ⇨ 36
  ```


- MAX(*expr*)

  Returnează valoarea maximă a argumentului *expr* pentru toate valorile diferite de NULL din rândurile selectate. Funcția MAX() se mai poate folosi și cu șiruri, caz în care returnează valoarea cea mai mare din punct de vedere lexicografic.

  ```
  SELECT MAX(coloana_mea) FROM tabelul_meu       ⇨ 9
  ```

- MIN(*expr*)

  Returnează valoarea minimă a argumentului *expr* pentru toate valorile diferite de NULL din rândurile selectate. Funcția MIN() se mai poate folosi și cu șiruri, caz în care returnează valoarea cea mai mică din punct de vedere lexicografic.

  ```
  SELECT MIN(coloana_mea) FROM tabelul_meu       ⇨ 1
  ```

- STD(*expr*)

  Returnează abaterea standard a argumentului *expr* pentru toate valorile diferite de NULL din rândurile selectate.

  ```
  SELECT STD(coloana_mea) FROM tabelul_meu       ⇨ 2.8284
  ```

- STDEV(*expr*)

  Această funcție este sinonimă cu STD().

- SUM(*expr*)

  Returnează suma argumentelor *expr* pentru toate valorile diferite de NULL din rândurile selectate.

  ```
  SELECT SUM(coloana_mea) FROM tabelul_meu       ⇨ 25
  ```

Funcții diverse

Funcțiile din această secțiune nu se încadrează în nici una din categoriile anterioare.

- BENCHMARK(*contor*,*expr*)

  Evaluează în mod repetat expresia *expr* de *contor* ori. BENCHMARK() este o funcție oarecum neobișnuită, în sensul că este destinată utilizării în programul client mysql. Valoarea returnată este întotdeauna 0, deci nu este de nici un folos. Valoarea de interes o constituie timpul scurs între momentul afișării rezultatului interogării și momentul afișării valorii returnate:

  ```
  mysql> SELECT BENCHMARK(100000,PASSWORD("secret"));
  +--------------------------------------+
  | BENCHMARK(100000,PASSWORD("secret")) |
  +--------------------------------------+
  |                                    0 |
  +--------------------------------------+
  1 row in set (1.63 sec)
  ```

  Timpul scurs reprezintă numai un indicator aproximativ al vitezei de evaluare a expresiei de către server, deoarece acesta reprezintă timpul măsurat „pe ceas" al clientului, nu timpul microprocesorului din server. Intervalul de timp poate fi influențat de factori precum încărcarea serverului, dacă serverul este într-o stare executabilă sau dacă este oprit la sosirea interogării etc. Va trebui să executați o interogare de mai multe ori, pentru a determina o valoare reprezentativă.

  Funcția BENCHMARK() a fost introdusă în MySQL versiunea 3.22.15.

- `BIT_COUNT(n)`

  Returnează numărul de biți activați din argument, acesta fiind tratat ca o valoare `BIGINT` (întreg pe 64 de biți).

```
BIT_COUNT(0)        ⇨ 0
BIT_COUNT(1)        ⇨ 1
BIT_COUNT(2)        ⇨ 1
BIT_COUNT(7)        ⇨ 3
BIT_COUNT(-1)       ⇨ 64
```

- `DATABASE()`

  Returnează un șir care conține numele bazei de date curente, respectiv un șir vid dacă nu există nici o bază de date curentă.

```
DATABASE()          ⇨ "samp_db"
```

- `DECODE(sir,parola)`

  Dacă este dat un șir criptat *sir*, obținut ca rezultat al unui apel la funcția `DECODE()`, funcția îl decriptează folosind șirul parolă *parola* și returnează șirul rezultant.

```
DECODE(ENCODE("secret","amestec"),"amestec")    ⇨ "secret"
```

- `ENCODE(sir,parola)`

  Criptează șirul *sir* folosind șirul parolă *parola* și returnează rezultatul sub formă de șir binar. Șirul poate fi decodificat folosind funcția `DECODE`. Deoarece rezultatul criptat este un șir binar, trebuie să folosiți o coloană care este de unul din tipurile `BLOB` dacă doriți să stocați șirul.

- `ENCRYPT(sir)`

  `ENCRYPT(sir,salt)`

  Criptează șirul *sir* și returnează șirul rezultant. Aceasta este o criptare ireversibilă. Argumentul *salt*, dacă este dat, trebuie să fie un șir format din două caractere. Dacă se specifică o valoare *salt*, rezultatul criptării șirului *sir* va fi de fiecare dată același. Dacă nu se specifică argumentul *salt*, apeluri identice la funcția `ENCRYPT()` vor determina în timp rezultate diferite.

```
ENCRYPT("secret","AB")    ⇨ "ABS5SGh1EL6bk"
ENCRYPT("secret","AB")    ⇨ "ABS5SGh1EL6bk"
ENCRYPT("secret")         ⇨ "z1oPSN18WRwFA"
ENCRYPT("secret")         ⇨ "12FJgqDtVOg7Q"
```

  Funcția `ENCRYPT()` folosește apelul la funcția sistem `crypt()` din UNIX și depinde de aceasta în două moduri. Mai întâi, dacă funcția `crypt()` nu este disponibilă în sistemul dumneavoastră, `ENCRYPT()` va returna întotdeauna `NULL`. În al doilea rând, `ENCRYPT()` depinde de modul în care `crypt()` operează în sistemele unde este prezentă. În particular, pe unele sisteme `crypt()` examinează numai primele opt caractere ale șirului care va fi criptat.

  Funcția `ENCRYPT()` a fost introdusă în versiunea MySQL 3.21.12. Începând de la versiunea MySQL 3.22.16, argumentul `salt` poate avea mai mult de două caractere lungime.

- `GET_LOCK(sir,expira)`

  Funcția `GET_LOCK()` este folosită în conjuncție cu funcția `RELEASE_LOCK()` pentru a executa blocarea consultativă (cooperativă). Puteți folosi cele două funcții pentru a scrie aplicații care cooperează în funcție de starea unui nume de blocare convenit în prealabil.

  `GET_LOCK()` este apelată cu un nume de blocare indicat de șirul `sir` și o valoare de expirare egală cu `expira` secunde. Funcția returnează 1 dacă blocarea a fost obținută cu succes în intervalul de expirare, 0 dacă încercarea de blocare a eșuat datorită expirării intervalului, respectiv `NULL` dacă s-a produs vreo eroare. Valoarea `expira` determină intervalul de tip alocat încercării de a obține o blocare, nu durata blocării. După ce a fost obținută, blocarea rămâne în vigoare până la anulare.

  Apelul următor obține o blocare denumită `"Aneta"` și așteaptă maximum 10 secunde pentru aceasta:

```
GET_LOCK("Aneta", 10)
```

  Blocarea se aplică numai șirului însuși. Nu blochează o bază de date, un tabel sau vreun rând sau o coloană dintr-un tabel. Cu alte cuvinte, blocarea nu împiedică nici un client să execute vreo operație cu tabelele din baza de date, motiv pentru care blocarea cu `GET_LOCK()` are numai un caracter consultativ – nu face decât să permită altor clienți care cooperează să determine dacă blocarea este sau nu în vigoare.

  Un client care dispune de o blocare asupra unui nume blochează încercările altor clienți de a bloca numele (sau încercările executate de alte fire din cadrul unui client multi-fir care folosește mai multe conexiuni cu serverul). Să presupunem că clientul 1 blochează șirul `"Aneta"`. Dacă clientul 2 încearcă să blocheze același șir, va fi blocat până când clientul 1 anulează blocarea sau până la scurgerea perioadei de expirare. În primul caz, clientul 2 va reuși să preia blocarea; în al doilea caz, nu va reuși.

  Deoarece doi clienți nu pot bloca un șir dat în același timp, aplicațiile care convin asupra unui nume pot folosi starea de blocare a numelui respectiv ca indicator privind oportunitatea execuției de operații relative la numele respectiv. De exemplu, puteți construi un nume de blocare bazat pe o valoare a unei chei unice a unui rând dintr-un tabel, pentru a permite blocarea cooperativă a rândului respectiv.

  O blocare este anulată în mod explicit prin apelarea funcției `RELEASE_LOCK()` cu numele blocării:

```
RELEASE_LOCK("Aneta")
```

  Funcția `RELEASE_LOCK()` returnează 1 dacă blocarea a fost eliberată cu succes, 0 dacă blocarea era deținută de o altă conexiune (vă puteți anula numai propriile dumneavoastră blocări), respectiv `NULL` dacă blocarea respectivă nu există.

  Orice blocare deținută de o conexiune client este de asemenea anulată automat la terminarea conexiunii, respectiv dacă clientul emite un alt apel la funcția `GET_LOCK()` (o conexiune client poate bloca un singur șir la un moment dat). În ultimul caz, blocarea deținută este anulată înainte de obținerea noii blocări, chiar dacă numele blocării este același.

  Funcția `GETLOCK(sir,0)` se poate folosi ca interogare ciclică pentru a determina, fără așteptare, dacă există sau nu o blocare a șirului `sir`. (Evident, șirul se va bloca în cazul în care nu este blocat, deci nu uitați să apelați funcția `RELEASE_LOCK()` dacă se impune acest lucru.)

  Funcția `GET_LOCK()` a fost introdusă în MySQL versiunea 3.21.7.

- LAST_INSERT_ID()

  LAST_INSERT_ID(*expr*)

  Fără nici un argument, funcția returnează valoarea AUTO_INCREMENT cel mai recent generată în timpul sesiunii curente a serverului, respectiv 0 dacă nu a fost generată o asemenea valoare. Cu un argument, LAST_INSERT_ID() este destinată utilizării într-o instrucțiune UPDATE. Rezultatul este tratat în același mod ca și o valoare generată automat, care este utilă pentru generarea secvențelor.

  Mai multe detalii despre această funcție se pot găsi în capitolul 2, „Lucrul cu date în MySQL". Pentru ambele forme ale funcției LAST_INSERT_ID(), valoarea este menținută de server pentru fiecare conexiune în parte și nu va fi modificată de alți clienți, nici chiar de acei clienți care determină crearea de noi valori generate automat.

  Forma cu un singur argument a funcției LAST_INSERT_ID() a fost introdusă în versiunea MySQL 3.22.9.

- LOAD_FILE(*nume_fisier*)

  Citește fișierul *nume_fisier* și returnează conținutul sub formă de șir. Fișierul trebuie să fie localizat pe server, trebuie specificată calea sa de acces absolută (completă) și trebuie să fie universal accesibil la citire, pentru a vă asigura că nu încercați să citiți un fișier protejat.

  Deoarece fișierul trebuie să se găsească pe server, trebuie să aveți privilegiul FILE. Dacă una din aceste condiții nu este îndeplinită, funcția LOAD_FILE() returnează NULL.

  Funcția LOAD_FILE() a fost introdusă în versiunea MySQL 3.23.0.

- MD5(*sir*)

  Calculează o sumă de control a șirului *sir*, bazată pe algoritmul MD5 Message-Digest al firmei RSA Data Security, Inc. Valoarea returnată este un șir alcătuit din 32 de cifre hexazecimale.

  MD5("secret") ⇨ "5ebe2294ecd0e0f08eab7690d2a6ee69"

  Funcția MD5() a fost introdusă în MySQL versiunea 3.23.2.

- PASSWORD(*sir*)

  Dacă este dat un șir *sir*, calculează și returnează un șir parolă criptat, de forma folosită în tabelele de acordare din MySQL. Aceasta este o criptare ireversibilă.

  PASSWORD("secret") ⇨ "428567f408994404"

  Rețineți că funcția PASSWORD() *nu* folosește același algoritm ca și cel utilizat în UNIX pentru criptarea parolelor conturilor de utilizator. Pentru acel tip de criptare, folosiți funcția ENCRYPT().

- RELEASE_LOCK(*sir*)

  RELEASE_LOCK() este folosită în conjuncție cu GET_LOCK(). Pentru detalii, consultați descrierea funcției GET_LOCK.

  Funcția RELEASE_LOCK() a fost introdusă în versiunea MySQL 3.21.7.

- SESSION_USER()

  Această funcție este sinonimă cu USER().

- SYSTEM_USER()

  Această funcție este sinonimă cu USER().

- USER()

  Returnează un șir care reprezintă numele de utilizator al clientului curent. Începân de la versiunea MySQL 3.22.1, șirul este de forma "utilizator@gazda", unde utilizator este numele utilizatorului și gazda este numele gazdei de la care clientul s-conectat la server.

  USER() ⇨ "paul@localhost"
  SUBSTRING_INDEX(USER(),"@",1) ⇨ "paul"
  SUBSTRING_INDEX(USER(),"@",-1) ⇨ "localhost"

- VERSION()

  Returnează un șir care descrie versiunea serverului, de exemplu "3.22.25-log". Sufixel care pot apărea după numărul versiunii serverului sunt -log (este activată caracteristic de consemnare în jurnal), -debug (serverul rulează în mod depanare) sau -demo (serveri rulează în modul demo).

  VERSION() ⇨ "3.23.1-alpha-log"

  Funcția VERSION()a fost introdusă în versiunea MySQL 3.21.13.

ANEXA D

Referință de sintaxă SQL

Această anexă descrie fiecare dintre instrucțiunile SQL furnizate de MySQL. De asemenea, este prezentată sintaxa pentru scrierea comentariilor în programele SQL. Comentariile sunt folosite pentru redactarea textelor având caracter descriptiv care sunt ignorate de server și pentru a ascunde cuvinte cheie specifice sistemului MySQL (aceste cuvinte cheie vor fi executate de MySQL, dar vor fi ignorate de alte servere de baze de date). Vezi secțiunea „Sintaxa comentariilor".

Dezvoltarea sistemului MySQL este perpetuă, deci îmbunătățirile aduse implementării sale a sistemului SQL se produc sistematic. Este util să consultați ocazional manualul de referință MySQL pe suport electronic, pentru a fi la curent cu facilitățile adăugate. Manualul este disponibil la adresa `http://www.mysql.com/`.

Descrierile sintactice folosesc următoarele convenții:

- Informațiile opționale sunt incluse între paranteze drepte ([ ]). Dacă o listă este delimitată de paranteze drepte, se poate selecta o singură alternativă.
- Dacă o listă este delimitată prin paranteze acolade ({ }), se impune selectarea unei alternative.
- Barele verticale (|) separă articolele alternative dintr-o listă.
- Notația cu puncte de suspensie (...) arată că articolul anterior punctelor de suspensie se poate repeta.

Dacă nu se specifică altceva, instrucțiunile enumerate aici sunt prezente în MySQL cel puțin de la versiunea MySQL 3.22.0.

Instrucțiuni SQL

Această secțiune descrie sintaxa și semnificația fiecăreia dintre instrucțiunile SQL folosite în sistemul MySQL. O instrucțiune va eșua dacă nu dispuneți de privilegiile necesare pentru a le executa. De exemplu, `USE nume_baza_de_date` va eșua dacă nu aveți permisiunea de a obține accesul la baza de date `nume_baza_de_date`.

ALTER TABLE

```
ALTER [IGNORE] TABLE nume_tabel lista_actiuni
```

Instrucțiunea `ALTER TABLE` vă permite să redenumiți tabele sau să le modificați structura. Pentru a o folosi, specificați numele tabelului `nume_tabel`, apoi indicați specificațiile pentru una sau mai multe operații care trebuie efectuate asupra tabelului. Cuvântul cheie

`IGNORE` intră în acțiune dacă în cheile unice ale noului tabel rezultă valori duplicate. Fără `IGNORE`, efectul instrucțiunii `ALTER TABLE` este anulat. Cu `IGNORE`, rândurile care conțin valori duplicate pentru chei unice vor fi șterse.

Parametrul `lista_actiuni` specifică una sau mai multe acțiuni, separate prin virgulă. Fiecare acțiune este executată pe rând. O acțiune poate fi oricare din următoarele:

- `ADD [COLUMN]` *`declaratie_coloana`* `[FIRST | AFTER` *`nume_coloana`*`]`

  Adaugă o coloană la tabel. *`declaratie_coloana`* reprezintă declarația coloanei; este în același format ca și cel folosit pentru instrucțiunea `CREATE TABLE`. Coloana devine prima coloană din tabel, dacă este dat cuvântul cheie `FIRST`, respectiv este plasată după coloana specificată, dacă este dată opțiunea `AFTER nume_coloana`. Dacă amplasamentul coloanei nu este specificat, coloana devine ultima coloană din tabel.

```
ALTER TABLE membru
    ADD membru_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
ALTER TABLE membru
    ADD membru_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
    FIRST
ALTER TABLE membru
    ADD membru_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
    AFTER sufix
```

- `ADD INDEX [`*`nume_index`*`] (`*`coloane_index`*`)`

  Adaugă un index la tabel. Indexul se bazează pe coloanele specificate în *`coloane_index`*, care trebuie să fie toate coloane în tabelul *`nume_tabel`*. Dacă sunt precizate mai multe coloane, trebuie să le separați prin virgulă. Pentru coloane de tip `CHAR` și `VARCHAR`, puteți indexa un prefix al coloanei, folosind sintaxa `nume_coloana(n)` pentru a indexa primele `n` caractere ale valorilor coloanei. Pentru coloanele `BLOB` și `TEXT`, *trebuie* să specificați o valoare a prefixului; nu puteți indexa întreaga coloană. Dacă numele indexului *`nume_index`* nu este specificat, atunci se va selecta automat un nume, în funcție de numele primei coloane indexate.

- `ADD PRIMARY KEY (`*`coloane_index`*`)`

  Adaugă o cheie primară la coloanele date. Cheia primește numele `PRIMARY`. Opțiunea *`coloane_index`* este specificată ca în cazul instrucțiunii `ADD INDEX`. Dacă o cheie primară există deja, se va produce o eroare.

```
ALTER TABLE presedinte ADD PRIMARY KEY (nume, prenume)
```

- `ADD UNIQUE [`*`nume_index`*`] (`*`coloane_index`*`)`

  Adaugă un index cu valori unice la *`nume_tabel`*. Opțiunile *`nume_index`* și *`coloane_index`* sunt specificate ca în cazul instrucțiunii `ADD INDEX`.

```
ALTER TABLE absente ADD UNIQUE id_data (elev_id, data)
```

- `ALTER [COLUMN]` *`nume_coloana`* `{SET DEFAULT` *`valoare`* `| DROP DEFAULT}`

  Modifică valoarea prestabilită a coloanei date, fie înlocuind-o cu valoarea specificată, fie prin ștergerea valorii prestabilite curente. În ultimul caz, se va atribui o nouă valoare prestabilită, așa cum se va arăta în cazul instrucțiunii `CREATE TABLE`.

```
ALTER TABLE eveniment ALTER tip SET DEFAULT "C"
ALTER TABLE eveniment ALTER tip DROP DEFAULT
```

- CHANGE [COLUMN] *nume_coloana declaratie_coloana*

  Modifică numele și definiția unei coloane. *nume_coloana* este numele curent al coloanei, iar *declaratie_coloana* este declarația coloanei în urma modificării. Opțiunea *declaratie_coloana* este în același format ca și cel folosit pentru instrucțiunea CREATE TABLE. Rețineți că declarația include noul nume al coloanei; dacă nu doriți să modificați numele, veți specifica același nume de două ori.

```
ALTER TABLE elev CHANGE nume nume VARCHAR(40)
ALTER TABLE elev CHANGE nume nume_elev CHAR(30) NOT NULL
```

- DROP [COLUMN] *nume_coloana*

  Elimină coloana dată din tabel. În cazul în care coloana face parte din vreun index, atunci va fi eliminată din indexul sau indexurile respective. Dacă vor fi eliminate toate coloanele dintr-un index, indexul va fi de asemenea eliminat.

```
ALTER TABLE presedinte DROP sufix
```

- DROP INDEX *nume_index*

  Elimină indexul dat din tabel.

```
ALTER TABLE membru DROP INDEX nume
```

- DROP PRIMARY KEY

  Elimină cheia primară din tabel. Dacă un tabel nu are nici un index unic care a fost creat drept PRIMARY KEY, dar are unul sau mai multe indexuri UNIQUE, primul din aceste indexuri este șters.

```
ALTER TABLE presedinte DROP PRIMARY KEY
```

- MODIFY [COLUMN] *declaratie_coloana*

  Modifică declarația unei coloane. Declarația de coloană *declaratie_coloana* este dată folosind același format ca și cel pentru descrierile de coloană din cazul instrucțiunii CREATE TABLE. Declarația începe cu un nume de coloană, unde se indică modul în care trebuie modificată coloana. Instrucțiunea MODIFY a fost introdusă în MySQL versiunea 3.22.16.

```
ALTER TABLE elev MODIFY nume VARCHAR(40) DEFAULT "" NOT NULL
```

- RENAME [AS] *nume_tabel_nou*

  Modifică numele tabelului *nume_tabel* în *nume_tabel_nou*.

```
ALTER TABLE presedinte RENAME prezident
```

- *optiuni_tabel*

  Specifică opțiunile pentru tabel din categoria care poate fi precizată în partea *optiuni_tabel* a unei instrucțiuni CREATE TABLE.

```
ALTER TABLE puncte TYPE = MYISAM CHECKSUM = 1
```

CREATE DATABASE

```
CREATE_DATABASE nume_baza_de_date
```

Creează o bază de date cu numele dat. Instrucțiunea eșuează dacă baza de date există deja sau dacă nu dispuneți de privilegiul adecvat pentru a o crea.

CREATE FUNCTION

```
CREATE [AGGREGATE] FUNCTION nume_functie
    RETURNS {STRING | REAL | INTEGER}
    SONAME nume_bibl_partajata
```

Specifică o funcție definită de utilizator (*user-defined function* - UDF) care va fi încărcată în tabelul func din baza de date mysql. *nume_functie* este numele sub care doriți să fie desemnată funcția în instrucțiunile SQL. Cuvântul cheie care urmează după RETURNS indică tipul valorii returnate de funcție. *nume_bibl_partajata* este calea de acces a fișierului care conține codul executabil sau funcția.

Cuvântul cheie AGGREGATE arată că funcția este o funcție de grupare, precum SUM() sau MAX(). AGGREGATE a fost introdus în versiunea MySQL 3.23.5.

CREATE FUNCTION impune ca serverul să fie construit ca binar cu legături dinamice (nu ca binar static), deoarece mecanismul UDF impune legarea dinamică. Pentru instrucțiuni referitoare la scrierea funcțiilor definite de utilizator, examinați manualul de referință MySQL.

CREATE INDEX

```
CREATE [UNIQUE] INDEX nume_index ON nume_tabel (coloane_index)
```

Adaugă un index cu numele *nume_index* la tabelul *nume_tabel*. Această instrucțiune este tratată ca și instrucțiunile ALTER TABLE ADD INDEX sau ALTER TABLE ADD UNIQUE, în conformitate cu absența sau prezența cuvântului cheie UNIQUE. Pentru detalii, examinați informațiile aferente instrucțiunii ALTER TABLE. Funcția CREATE INDEX a fost introdusă în versiunea MySQL 3.22.

CREATE TABLE

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nume_tabel
    (creare_definitie,...)
    [optiuni_tabel]
    [[IGNORE | REPLACE] instructiune_select]

creare_definitie:
    { declaratie_coloana
        | PRIMARY KEY (coloane_index)
        | KEY [nume_index] [coloane_index]
        | INDEX [nume_index] [coloane_index]
        | UNIQUE INDEX [nume_index] [coloane_index]
        | [CONSTRAINT simbol] FOREIGN KEY nume_index (coloane_index)
        | [definitie_referinta]
        | CHECK (expr) }
```

```
declaratie_coloana:
    nume_coloana tip_coloana
        [NOT NULL | NULL] [DEFAULT valoare_prestabilita]
        [AUTO_INCREMENT] [PRIMARY KEY] [definitie_referinta]

definitie_referinta:
    REFERENCES nume_tabel [(coloane_index)]
        [MATCH FULL | MATCH PARTIAL]
        [ON DELETE optiune_referinta]
        [ON UPDATE optiune_referinta]

optiune_referinta:
    {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}
```

Instrucțiunea CREATE TABLE creează un tabel nou, denumit *nume_tabel*. Dacă este dat cuvântul cheie TEMPORARY, tabelul există numai până la terminarea, în condiții normale sau anormale, a conexiunii client curente, respectiv până la emiterea unei instrucțiuni DROP TABLE. Un tabel temporar este vizibil numai pentru clientul care l-a creat.

În mod normal, încercarea de a crea un tabel cu un nume care există deja va avea ca efect o eroare. Există două situații în care nu se vor produce erori. Mai întâi, dacă este specificată clauza IF NOT EXISTS, tabelul nu este creat, dar nu se produce nici o eroare. A doua situație: dacă este specificată opțiunea TEMPORARY și tabelul original nu este un tabel temporar, atunci va fi creat noul tabel temporar, dar tabelul original denumit *nume_tabel* devine ascuns clientului pe durata existenței tabelului temporar. Tabelul original rămâne vizibil pentru ceilalți clienți. Tabelul original redevine vizibil din nou fie la următoarea sesiune client, fie dacă este emisă o instrucțiune DROP TABLE explicită pentru tabelul temporar.

Lista *creare_definitie* denumește coloanele și indexurile pe care doriți să le creați. Lista este opțională dacă creați tabelul prin intermediul unei instrucțiuni SELECT finale. Clauza *optiuni_tabel* vă permite să specificați diferite opțiuni pentru tabel. Dacă este specificată o instrucțiune finală *instructiune_select* (sub forma unei instrucțiuni SELECT arbitrare), tabelul este creat folosind setul de rezultate returnat de instrucțiunea SELECT. Aceste clauze vor fi descrise mai detaliat în secțiunile următoare.

Cuvântul cheie TEMPORARY, clauza IF NOT EXISTS, clauza optiuni_tabel și capacitatea de a crea un tabel pornind de la rezultatul unei instrucțiuni SELECT au fost introduse în versiunea MySQL 3.23.

O opțiune *creare_definitie* poate fi o declarație de coloană, o definiție de index, o clauză FOREIGN KEY, o opțiune *referinta_definitie* sau o clauză CHECK. Ultimele trei sunt analizate pentru a se determina compatibilitatea cu alte sisteme de baze de date, dar de regulă sunt ignorate.

O declarație de coloană *declaratie_coloana* începe cu un nume de coloană *nume_coloana* și de un tip *tip_coloana*, după care pot urma numeroase cuvinte cheie opționale. Tipul de coloană poate fi oricare dintre tipurile prezentate în Anexa B, „Referință de tipuri de coloane". Pentru a determina atributele specifice unui tip anume, atribute aplicabile coloanelor pe care doriți să le declarați, examinați anexa respectivă. Cuvintele cheie opționale care pot apărea după numele coloanei sunt următoarele:

- NULL sau NOT NULL

  Arată că respectiva coloană poate conține sau nu valori NULL. Dacă nici una din aceste opțiuni nu este specificată, NULL este opțiunea prestabilită.

- DEFAULT valoare_prestabilita

  Indică valoarea prestabilită a coloanei. Nu se poate folosi pentru tipuri BLOB sau TEXT. Dacă nu este specificată nici o valoare prestabilită, atunci va fi atribuită o asemenea valoare. Pentru coloane care pot accepta valori NULL, valoarea prestabilită este NULL. Pentru coloanele care nu pot accepta valori NULL, valoarea prestabilită este atribuită după cum urmează:

  - Pentru coloanele numerice, valoarea prestabilită este 0, cu excepția coloanelor AUTO_INCREMENT. Pentru AUTO_INCREMENT, valoarea prestabilită este următorul număr din secvența de numere aferentă coloanei.
  - Pentru tipuri dată-oră altele decât TIMESTAMP, valoarea prestabilită este valoarea „zero" aferentă tipului respectiv (de exemplu "0000-00-00" pentru tipul DATE). Pentru TIMESTAMP, valoarea prestabilită este data și ora curentă.
  - Pentru tipuri șir, altele decât ENUM, valoarea prestabilită este șirul vid. Pentru ENUM, valoarea prestabilită este primul element din enumerare.

- AUTO_INCREMENT

  Acest cuvânt cheie se aplică numai tipurilor de coloane cu valori întregi. O coloană AUTO_INCREMENT este mai deosebită, în sensul că atunci când introduceți NULL într-o asemenea coloană, valoarea inserată efectiv este cu o unitate mai mare decât valoarea curentă maximă din coloană. Valorile AUTO_INCREMENT pornesc în mod prestabilit de la 1; prima valoare poate fi specificată în mod explicit cu opțiunea de tabel AUTO_INCREMENT. De asemenea, coloanele trebuie specificate sub formă de index UNIQUE sau PRIMARY KEY și trebuie să fie de tipul NOT NULL. Într-un tabel poate exista cel mult o coloană AUTO_INCREMENT.

- PRIMARY KEY

  Arată că respectiva coloană este de tip PRIMARY KEY.

- UNIQUE

  Arată că respectiva coloană este un index UNIQUE. Acest atribut poate fi specificat începând de la versiunea MySQL 3.23.

Clauzele PRIMARY KEY, UNIQUE, INDEX și KEY specifică indexuri; PRIMARY KEY și UNIQUE specifică indexuri care trebuie să conțină valori unice. INDEX și KEY sunt sinonime și specifică indexuri care pot conține valori duplicate. Indexul este bazat pe coloanele denumite în *coloane_index*, care trebuie să fie coloane în tabelul *nume_tabel*. Dacă sunt precizate mai multe coloane, acestea trebuie separate prin virgule. Pentru coloanele CHAR și VARCHAR, puteți indexa un prefix al coloanei, folosind sintaxa *nume_coloana(n)* pentru a indexa primele n caractere ale valorilor din coloane. Pentru coloanele BLOB și TEXT, *trebuie* să specificați o valoare a prefixului; nu puteți indexa întreaga coloană. Dacă numele indexului *nume_index* nu este specificat, atunci se va selecta un nume în mod automat, în funcție de numele primei coloane indexate.

Anterior versiunii MySQL 3.23, coloanele indexate trebuie declarate de tip `NOT NULL`. Începând de la versiunea MySQL 3.23, numai coloanele din indexurile `PRIMARY KEY` trebuie declarate `NOT NULL`.

Clauza *`optiuni_tabel`* este disponibilă începând de la versiunea MySQL 3.23. În prezent, poate conține una sau mai multe dintre opțiunile din lista următoare. Dacă există mai multe opțiuni, acestea nu trebuie separate prin virgule. Fiecare specificator se aplică tuturor tipurilor de tabele, cu excepția situațiilor când există alte precizări.

- `AUTO_INCREMENT = n`

  Prima valoare `AUTO_INCREMENT` care va fi generată pentru tabel (valabil numai pentru tabelele MyISAM).

- `AVG_ROW_LENGTH = n`

  Lungimea medie aproximativă a rândurilor tabelului dumneavoastră. Această opțiune este folosită de către MySQL pentru a determina dimensiunea maximă a fișierului de date și este necesară numai pentru tabele care conțin coloane `BLOB` sau `TEXT`, când tabelele respective pot deveni mai mari de 4GB.

- `CHECKSUM = {0 | 1}`

  Dacă această opțiune are valoarea 1, MySQL folosește o sumă de control pentru fiecare rând de tabel. Există un mic dezavantaj pentru actualizările aduse tabelului, dar astfel se îmbunătățește procesul de verificare a tabelului (valabil numai pentru tabelele MyISAM).

- `COMMENT = "sir"`

  Un comentariu pentru tabel. Lungimea maximă este de 60 de caractere.

- `DELAY_KEY_WRITE = {0 | 1}`

  Dacă această opțiune are valoarea 1, memoria cache a indexului este golită numai ocazional pentru tabel, nu după fiecare operație de inserție (valabil numai pentru tabelele MyISAM).

- `MAX_ROWS = n`

  Numărul maxim de rânduri pe care intenționați să-l stocați în tabel.

- `MIN_ROWS = n`

  Numărul minim de rânduri pe care intenționați să-l stocați în tabel.

- `PACK_KEYS = {0 | 1}`

  Dacă această opțiune este configurată la valoarea 1, blocurile de index sunt împachetate la un procentaj mai mare decât cel normal. Efectul normal este o penalizare la actualizare și o îmbunătățire a performanțelor operațiilor de regăsire. Dacă opțiunea are valoarea 0, blocurile de index nu sunt împachetate în mod special. În acest caz, sunt împachetate numai indexurile cu valori cheie de tip `CHAR` sau `VARCHAR` cu o lungime de minimum 8 caractere (valabil numai pentru tabelele MyISAM și ISAM).

- `PASSWORD = "sir"`

  Specifică o parolă pentru criptarea fișierului de descriere a tabelului. În mod normal, această operație nu are nici un efect; opțiunea este activată numai pentru anumiți clienți care beneficiază de un contract de asistență.

- `TYPE = {ISAM | MYISAM | HEAP}`

  Specifică formatul de stocare în tabel. Caracteristicile acestor formate de stocare sunt descrise în secțiunea „Specificatori ai tipului de stocare în tabel" din capitolul 3, „Sintaxa și utilizarea SQL în MySQL". Formatul prestabilit pentru MySQL începând de la versiunea 3.23 este MyISAM. Anterior versiunii MySQL 3.23, instrucțiunea `CREATE TABLE` crea întotdeauna tabele în formatul ISAM.

Dacă este specificată o clauză *`instructiune_select`* (sub forma unei interogări `SELECT`), tabelul este creat folosind conținutul setului de rezultate returnat de interogare. Rândurile care reproduc valorile dintr-un index unic sunt fie ignorate, fie înlocuiesc rândurile existente, în funcție de specificarea uneia dintre opțiunile `IGNORE` sau `REPLACE`. Dacă nu este specificată nici una dintre aceste opțiuni, se va produce o eroare, iar toate înregistrările rămase sunt ignorate.

Să creăm un tabel cu trei coloane. Coloana `id` este de tip `PRIMARY KEY`, iar cele două coloane de nume sunt indexate împreună:

```
CREATE TABLE client
(
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    nume CHAR(30) NOT NULL,
    prenume CHAR(20) NOT NULL,
    PRIMARY KEY (id),
    INDEX (nume, prenume)
)
```

Creați un tabel temporar și transformați-l în tabel HEAP (stocat în memorie), pentru un acces mai rapid:

```
CREATE TEMPORARY TABLE tmp_tabel
    (id MEDIUMINT NOT NULL UNIQUE, nume CHAR(40))
    TYPE = HEAP
```

Creați un tabel ca o copie a unui alt tabel:

```
CREATE TABLE prez_copie SELECT * FROM presedinte
```

Creați un tabel folosind numai o parte a unui alt tabel:

```
CREATE TABLE prez_traiesc SELECT nume, prenume, nastere
    FROM presedinte WHERE deces IS NULL
```

Dacă declarațiile de creare sunt specificate pentru un tabel creat și populat prin intermediul unei instrucțiuni `SELECT` finale, declarațiile sunt aplicate după ce conținutul tabelului a fost inserat în tabel. De exemplu, puteți declara că o coloană selectată trebuie transformată în `PRIMARY KEY`:

```
CREATE TABLE tabel_nou (PRIMARY KEY(a)) SELECT a, b, c FROM tabel_vechi
```

DELETE

```
DELETE [LOW_PRIORITY] FROM nume_tabel
    [WHERE unde_expr] [LIMIT n]
```

Şterge rânduri din tabelul *nume_tabel*. Rândurile şterse sunt cele care corespund condiţiilor specificate în expresia *unde_expr*:

```
DELETE FROM puncte WHERE eveniment_id = 14
DELETE FROM membru WHERE expirare < CURRENT_DATE
```

În cazul în care clauza WHERE este omisă, *toate înregistrările din tabel vor fi şterse.*

Instrucţiunea DELETE returnează numărul de înregistrări şterse, cu excepţia situaţiei când ştergeţi toate înregistrările prin nespecificarea clauzei WHERE. MySQL optimizează acest caz special prin re-crearea fişierelor de date şi a fişierelor index pornind de la zero, nu prin ştergerea rând cu rând a înregistrărilor. Această operaţie este extrem de rapidă, dar nu pune la dispoziţie numărul rândurilor afectate şi returnează valoarea zero. Pentru a obţine un număr autentic, specificaţi o clauză WHERE care returnează toate înregistrările. De exemplu:

```
DELETE FROM nume_tabel WHERE 1 > 0
```

Totuşi, apare o diminuare sensibilă a performanţelor.

Dacă este specificată opţiunea LOW_PRIORITY, execuţia instrucţiunii este amânată până când nici un client nu mai citeşte din tabel. Opţiunea LOW_PRIORITY a fost introdusă în MySQL versiunea 3.22.5.

Dacă este dată clauza LIMIT, valoarea n specifică numărul maxim de rânduri care vor fi şterse. Clauza LIMIT a fost introdusă în MySQL versiunea 3.22.7.

DESCRIBE

```
{DESCRIBE | DESC} nume_tabel {nume_coloana | model}
```

Această instrucţiune furnizează aceleaşi informaţii ca şi SHOW COLUMNS. Pentru mai multe informaţii, consultaţi descrierea instrucţiunii SHOW.

DROP DATABASE

```
DROP DATABASE [IF EXISTS] nume_baza_de_date
```

Elimină (şterge) baza de date specificată. După ce aţi şters o bază de date, aceasta a dispărut pe vecie, deci fiţi atent. Instrucţiunea eşuează dacă baza de date nu există (cu excepţia cazului când specificaţi IF EXISTS) sau dacă nu aveţi privilegiul adecvat. Opţiunea IF EXISTS a fost introdusă în versiunea MySQL 3.22.2.

O bază de date este reprezentată într-un catalog amplasat în catalogul de date. Dacă în catalogul respectiv aţi inserat fişiere care nu reprezintă tabele, fişierele respective nu vor fi şterse de instrucţiunea DROP DATABASE, caz în care nici catalogul bazei de date nu va fi eliminat.

DROP FUNCTION

```
DROP FUNCTION nume_functie
```

Elimină o funcţie definită de utilizator care a fost încărcată anterior cu instrucţiunea CREATE FUNCTION.

DROP INDEX

```
DROP INDEX nume_index ON nume_tabel
```

Şterge indexul *nume_index* din tabelul *nume_tabel*. Această instrucţiune este manipulată asemenea unei instrucţiuni ALTER TABLE DROP INDEX. Pentru detalii, examinaţi descrierea instrucţiunii ALTER TABLE. Instrucţiunea DROP INDEX a fost introdusă în MySQL versiunea 3.22.

DROP TABLE

```
DROP TABLE [IF EXISTS] nume_tabel [, nume_tabel] ...
```

Şterge tabelul sau tabelele specificate din baza de date căreia îi aparţin acestea. Dacă este dată clauza IF EXISTS, ştergerea unui tabel inexistent nu este o eroare. Clauza IF EXISTS a fost introdusă în MySQL versiunea 3.22.2.

EXPLAIN

```
EXPLAIN nume_tabel
EXPLAIN instructiune_select
```

Prima formă a acestei instrucţiuni este echivalentă cu SHOW COLUMNS FROM *nume_tabel*. Pentru mai multe informaţii, examinaţi descrierea instrucţiunii SHOW.

Cea de-a doua formă a instrucţiunii EXPLAIN furnizează informaţii referitoare la modul în care MySQL va executa instrucţiunea SELECT care urmează după cuvântul cheie EXPLAIN.

```
EXPLAIN SELECT puncte.* FROM puncte, eveniment
    WHERE puncte.eveniment_id = eveniment.eveniment_id
    AND eveniment.eveniment_id = 14
```

Datele de ieşire ale instrucţiunii EXPLAIN sunt alcătuite din unul sau mai multe rânduri, care conţin următoarele coloane:

- `table`

  Tabelul la care se referă rândul de ieşire.

- `type`

  Tipul de unire pe care îl va efectua MySQL. Tipurile posibile sunt următoarele, în ordine descrescătoare a calităţii: system, const, eq_ref, ref, range, index şi ALL. Tipurile mai bune sunt mai restrictive, ceea ce înseamnă că MySQL trebuie să examineze mai puţine rânduri din tabel atunci când execută operaţia de regăsire.

- `possible_keys`

  Indexurile pe care MySQL le consideră adecvate pentru găsirea rândurilor din tabelul denumit în coloana row. Valoarea NULL arată că nu au fost găsite indexuri.

- `key`

  Indexul pe care îl va folosi efectiv MySQL pentru găsirea rândurilor din tabel. Valoarea NULL arată că nu se va folosi nici un index.

- `key_len`

  Porţiunea din index care va fi folosită. Aceasta poate fi mai mică decât lungimea totală a rândului de index, dacă MySQL va folosi un prefix de stânga al indexului.

- `ref`

  Valorile cu care MySQL va compara valorile din index. Cuvântul `const` sau expresia `???` arată că termenul de comparație este o constantă; un nume de coloană indică o comparație între coloane.
- `rows`

  O valoare estimativă a numărului de rânduri din tabel pe care MySQL trebuie să le examineze pentru a executa interogarea. Produsul valorilor din această coloană este o valoare aproximativă a numărului total de combinații de rânduri care trebuie examinate din toate tabelele.
- `Extra`

  Opțiunea `Only index` arată că MySQL poate regăsi informațiile aferente tabelului folosind numai informațiile din index, fără a examina fișierul de date. Opțiunea `where used` indică utilizarea informațiilor din clauza `WHERE` a instrucțiunii `SELECT`.

FLUSH

```
FLUSH optiune_golire [, optiune_golire] ...
```

Golește diferitele zone cache interne folosite de server. Opțiunea `optiune_golire` poate fi oricare dintre articolele din lista următoare:

- `HOSTS`

  Golește zona cache aferentă gazdelor.
- `LOGS`

  Golește fișierele jurnal, închizându-le și redeschizându-le.
- `PRIVILEGES`

  Reîncarcă tabelele de acordare.
- `STATUS`

  Reinițializează variabilele de stare.
- `TABLES`

  Închide toate tabelele deschise în zona cache a tabelului.

Instrucțiunea `FLUSH` a fost introdusă în MySQL 3.22.9. Opțiunea de golire `STATUS` a fost introdusă în MySQL versiunea 3.22.11.

GRANT

```
GRANT tip_priv [(lista_coloane)][, tip_priv [(lista_coloane)] ] ...
   ON {*.* | * | nume_db.* | nume_db.nume_tabel | nume_tabel}
   TO utilizator [, utilizator ] ...
   [WITH GRANT OPTION]
```

Instrucțiunea `GRANT` acordă privilegii de acces unuia sau mai multor utilizatori MySQL. Valoarea `tip_priv` specifică privilegiile care vor trebui acordate și constă din una sau mai multe tipuri de privilegii separate prin virgulă, selectate din următoarea listă:

Specificator de privilegiu	Operație permisă de privilegiu
`ALTER`	Modifică tabele și indexuri
`CREATE`	Creează baze de date și tabele
`DELETE`	Șterge din tabele înregistrări existente
`DROP`	Șterge (elimină) baze de date și tabele
`INDEX`	Creează sau șterge indexuri
`INSERT`	Inserează înregistrări noi în tabele
`REFERENCES`	Nefolosit
`SELECT`	Regăsește înregistrări existente din tabele
`UPDATE`	Modifică înregistrări existente din tabele
`FILE`	Citește și scrie fișiere pe server
`PROCESS`	Vizualizează informații despre firele care se execută în cadrul serverului sau suprimă firele
`RELOAD`	Reîncarcă tabelele de acordare sau golește jurnalele, zona cache aferentă gazdelor sau zona cache a tabelelor
`SHUTDOWN`	Oprește serverul
`ALL`	Orice acțiune; sinonim cu `ALL PRIVILEGES`
`USAGE`	Un privilegiu special „fără privilegii"

Vă puteți întotdeauna vizualiza sau suprima propriile fire. Privilegiul `PROCESS` vă permite să vizualizați sau să suprimați orice fir.

Clauza `ON` specifică domeniul de acordare a privilegiilor, așa cum se poate vedea în tabelul următor:

Specificator de privilegii	Nivelul de aplicare a privilegiilor
`*.*`	Privilegii globale; toate bazele de date, toate tabelele
`*`	Privilegii globale, dacă nu a fost selectată nici o bază de date prestabilită; privilegii la nivel de bază de date pentru baza de date curentă, în caz contrar
`nume_db.*`	Privilegii la nivel de bază de date; toate tabelele din baza de date specificată
`nume_db.nume_tabel`	Privilegii la nivel de tabel; toate coloanele din tabelul specificat
`nume_tabel`	Privilegii la nivel de tabel; toate coloanele din tabelul denumit în baza de date prestabilită

Când este specificat un tabel în clauza `ON`, privilegiile trebuie să fie la nivel de coloană; acest lucru se obține prin denumirea uneia sau mai multor coloane separate prin virgulă în clauza `lista_coloane`.

Clauza `TO` specifică unul sau mai mulți utilizatori cărora trebuie să li se acorde privilegiile. Fiecare `utilizator` constă dintr-un specificator în formatul *`nume_utilizator@nume_gazda`*, cu o clauză opțională `IDENTIFIED BY`. *`nume_utilizator`* poate fi un nume sau un șir vid (`""`); ultimul specifică un utilizator anonim. *`nume_gazda`* poate fi specificat ca `localhost`, ca un

nume de gazdă, ca o adresă IP sau ca un model care corespunde unui nume de domeniu sau unui număr de rețea. Caracterele dintr-un model sunt `%` și `_`, cu aceeași semnificație ca pentru operatorul `LIKE`. Un *nume_utilizator* specificat singur, fără nici un nume de gazdă, este echivalent cu notația *nume_utilizator*@%. Începând de la versiunea MySQL 3.23, `nume_utilizator` poate fi o pereche număr IP-mască de rețea scrise sub forma *n.n.n.n/m*, unde *n.n.n.n* indică adresa IP, iar *m* indică numărul de biți ce trebuie utilizați pentru numărul rețelei.

Clauza `IDENTIFIED BY`, dacă este dată, atribuie utilizatorului o parolă. Parola trebuie specificată în format text simplu, fără a folosi funcția `PASSWORD`, spre deosebire de modul în care sunt specificate parolele pentru instrucțiunea `SET PASSWORD`. Dacă utilizatorul există deja și opțiunea `IDENTIFIED BY` este specificată, noua parolă o va înlocui pe cea veche. Parola existentă rămâne, în caz contrar, nemodificată.

Instrucțiunea `GRANT` a fost introdusă în versiunea MySQL 3.22.11.

Instrucțiunile următoare exemplifică unele moduri de utilizare a instrucțiunii `GRANT`. Pentru alte exemple, consultați capitolul 11, „Administrarea generală a sistemului MySQL".

Creați un utilizator `paul` cu acces la toate tabelele din baza de date `samp_db` de la orice gazdă. Următoarele două instrucțiuni sunt echivalente, deoarece omiterea numelui gazdei în identificatorul `utilizator` este echivalentă cu specificarea caracterului `%`:

```
GRANT ALL ON samp_db.* TO paul IDENTIFIED BY "secret"
GRANT ALL ON samp_db.* TO paul@% IDENTIFIED BY "secret"
```

Creați un utilizator care dispune numai de privilegii de citire pentru tabelele din baza de date `menagerie`. Utilizatorul se poate conecta de la orice gazdă din domeniul `xyz.com`:

```
GRANT SELECT ON menagerie.* TO numai_ochi@%.xyz.com
   IDENTIFIED BY "rugina"
```

Creați un utilizator cu privilegii complete, dar numai pentru tabelul `membru` din baza de date `samp_db`. Utilizatorul se poate conecta de la o singură gazdă:

```
GRANT ALL ON samp_db.membru TO membru_mgr@boa.snake.net
   IDENTIFIED BY "gogoasa"
```

Creați un super-utilizator care poate face orice, inclusiv să acorde privilegii altor utilizatori, dar care trebuie să se conecteze de la gazda locală:

```
GRANT ALL ON *.* TO maresitare@localhost IDENTIFIED BY "homer"
   WITH GRANT OPTION
```

Creați un utilizator anonim al bazei de date `menagerie`, care se poate conecta de la gazda locală, fără parolă:

```
GRANT ALL ON menagerie.* TO ""@localhost
```

INSERT

```
INSERT [LOW PRIORITY | DELAYED] [IGNORE] [INTO]
   nume_tabel [(lista_coloane)] VALUES (expr [, expr] ...) [, (...)] ...
INSERT [LOW PRIORITY | DELAYED] [IGNORE] [INTO]
   nume_tabel [(lista_coloane)] SELECT ...
INSERT [LOW PRIORITY | DELAYED] [IGNORE] [INTO]
   nume_tabel SET nume_coloana=expr [,nume_coloana=expr] ...
```


Inserează rânduri într-un tabel `nume_tabel` existent și returnează numărul de rânduri inserate. Cuvântul cheie `INTO` este obligatoriu în versiunile anterioare versiunii MySQL 3.22.5.

`LOW_PRIORITY` determină amânarea execuției instrucțiunii până când nici un client nu mai citește din tabel. Opțiunea `LOW_PRIORITY` a fost introdusă în MySQL versiunea 3.22.5.

Opțiunea `DELAYED` determină plasarea rândurilor într-o coadă în vederea unei inserții ulterioare, iar instrucțiunea returnează imediat, astfel încât clientul își poate continua execuția, fără să mai aștepte. Totuși, în acest caz, funcția `LAST_INSERT_ID()` nu va returna valoarea `AUTO_INCREMENT` aferentă pentru orice coloană `AUTO_INCREMENT` din tabel. Inserțiile cu ajutorul opțiunii `DELAYED` au fost introduse în MySQL versiunea 3.22.15.

Dacă este specificată opțiunea `IGNORE`, rândurile care reproduc valorile cheilor unice din rândurile existente sunt ignorate. Dacă valorile duplicate apar fără `IGNORE`, se va produce o eroare și nu se mai inserează nici un rând. Opțiunea `IGNORE` a fost introdusă în MySQL 3.22.10.

Prima formă a funcției `INSERT` necesită o listă `VALUES()` care specifică toate valorile care vor fi inserate. Dacă nu este dată nici o listă de coloane, lista `VALUES()` trebuie să specifice o valoare pentru fiecare coloană din tabel. Dacă este dată o listă de coloane, alcătuită din unul sau mai multe nume de coloană, separate prin virgulă, în lista `VALUES()` trebuie specificată câte o valoare pentru fiecare coloană. Coloanele care nu sunt specificate în lista de coloane își primesc valorile lor prestabilite. Începând de la versiunea MySQL 3.22.5, se pot specifica mai multe liste cu valori, permițând inserția mai multor rânduri folosind o singură instrucțiune `INSERT`. Începând cu versiunea MySQL 3.23.3, lista `VALUES()` poate fi vidă, ceea ce are ca rezultat inserția unui rând pentru care fiecare coloană își primește valoarea prestabilită.

A doua formă a instrucțiunii `INSERT` regăsește înregistrările în conformitate cu instrucțiunea `SELECT` și le inserează în `nume_tabel`. Instrucțiunea `SELECT` trebuie să selecteze toate coloanele din `nume_tabel`, respectiv toate coloanele din `lista_coloane`, dacă se specifică o listă de coloane. Când este specificată o listă de coloane, toate coloanele care nu sunt specificate în listă își vor primi valorile lor prestabilite.

Cea de-a treia formă a instrucțiunii `INSERT`, disponibilă începând cu versiunea MySQL 3.22.10, inserează coloane denumite în clauza `SET` în valorile date de expresiile corespunzătoare. Coloanele care nu sunt denumite își primesc valorile prestabilite.

```
INSERT INTO absente (elev_id, data) VALUES(14,"1999-11-03"),(34,NOW())
INSERT INTO absente SET elev_id = 14, data = "1999-11-03"
INSERT INTO absente SET elev_id = 34, data = NOW()
INSERT INTO puncte (elev_id, puncte, eveniment_id)
   SELECT elev_id, 100 AS puncte, 15 AS eveniment_id FROM elev
```

KILL

```
KILL fir_id
```

Suprimă firul de server cu identificatorul de fir *fir_id*. Trebuie să aveți privilegiul `PROCESS` pentru a suprima firul, cu excepția situației când firul respectiv vă aparține.

Comanda `mysqladmin kill` execută aceeași operație, dar permite specificarea în linia de comandă a mai multor valori de identificatori de fir. Instrucțiunea `KILL` permite numai un singur identificator.

Această instrucțiune a fost introdusă în versiunea MySQL 3.22.9.

LOAD DATA

```
LOAD DATA [LOW PRIORITY] [LOCAL] INFILE 'nume_fisier'
    [IGNORE | REPLACE]
    INTO TABLE nume_tabel
    optiuni_import
    [IGNORE n LINES]
    [(lista_coloane)]
```

Instrucțiunea LOAD DATA citește înregistrări din fișierul *nume_fisier* și le încarcă în cantități mari în tabelul *nume_tabel*. Operația este mai rapidă decât INSERT.

LOW PRIORITY determină amânarea execuției instrucțiunii până când nici un client nu mai citește din tabel. Această opțiune a fost introdusă în versiunea MySQL 3.23.

În mod normal, fișierul este citit direct de serverul situat pe gazda serverului. În acest caz, trebuie să aveți privilegiul FILE și fișierul trebuie să se găsească fie în catalogul bazei de date prestabilite, fie trebuie să fie universal accesibil la citire. Dacă se specifică opțiunea LOCAL, clientul citește fișierul pe gazda clientului și trimite conținutul fișierului la server, prin rețea. În acest caz, privilegiul FILE nu este obligatoriu. Opțiunea LOCAL a devenit funcțională în versiunea MySQL 3.22.15.

Când opțiunea LOCAL nu este specificată, serverul găsește fișierul după cum urmează:

- Dacă *nume_fisier* este o cale de acces absolută, serverul caută fișierul începând de la catalogul rădăcină.
- Dacă *nume_fisier* este o cale de acces relativă, interpretarea depinde de faptul dacă numele conține sau nu o singură componentă. În caz afirmativ, serverul caută fișierul în catalogul bazei de date prestabilite. Dacă numele fișierului conține mai multe componente, serverul caută fișierul începând de la catalogul de date al serverului.

  Dacă opțiunea LOCAL este dată, numele fișierului este interpretat astfel:

  - Dacă *nume_fisier* este o cale de acces absolută, clientul caută fișierul începând de la catalogul rădăcină.
  - Dacă *nume_fisier* este o cale de acces relativă, clientul caută fișierul începând de la catalogul dumneavoastră curent.

Rândurile care conțin valori duplicate ale unui index unic sunt fie ignorate, fie înlocuiesc rândurile existente, în funcție de specificarea opțiunii IGNORE sau REPLACE. Dacă nici una dintre aceste opțiuni nu este specificată, se va produce o eroare și toate înregistrările rămase vor fi ignorate.

Clauza *optiuni_import* indică formatul datelor. Opțiunile disponibile în această clauză se aplică și clauzei *optiuni_export* aferente instrucțiunii SELECT ... INTO OUTFILE. Sintaxa pentru *optiuni_import* este următoarea:

```
[FIELDS
    [TERMINATED BY 'sir']
    [[OPTIONALLY] ENCLOSED BY 'caracter']
    [ESCAPED BY 'caracter']]
  [LINES TERMINATED BY 'sir']
```


Valorile *sir* și *caracter* pot include următoarele secvențe escape pentru indicarea caracterelor speciale:

Secvență	Semnificație
\0	ASCII 0
\b	Ștergere caracter anterior
\n	Caracter linie nouă
\r	Retur de car
\s	Spațiu
\t	Caracter de tabulare
\'	Ghilimele simple
\"	Ghilimele duble
\\	Backslash

Începând de la versiunea MySQL 3.22.10, mai puteți folosi și constante hexazecimale pentru a indica anumite caractere arbitrare. De exemplu, LINES TERMINATED BY 0x02 arată că liniile se încheie prin caractere Ctrl-B (ASCII 2).

Dacă este precizată opțiunea FIELDS, atunci trebuie dată cel puțin una dintre clauzele TERMINATED BY, ENCLOSED BY sau ESCAPED BY, iar clauzele date trebuie specificate în ordinea indicată. Dacă opțiunile FIELDS și LINES sunt ambele indicate, opțiunea FIELDS trebuie să preceadă opțiunea LINES. Componentele clauzei FIELDS sunt folosite după cum urmează:

- TERMINATED BY arată caracterul sau caracterele care delimitează valorile din cadrul unei linii.
- ENCLOSED BY precizează un caracter de citare care este eliminat de la extremitățile valorilor câmpurilor, dacă este găsit. Această operație are loc indiferent dacă clauza OPTIONALLY este prezentă sau nu. Pentru ieșire (SELECT ... INTO OUTFILE), caracterul ENCLOSED BY este folosit pentru a încadra valorile câmpurilor în liniile de ieșire. Dacă este dată clauza OPTIONALLY, valorile sunt încadrate între ghilimele numai pentru coloanele CHAR și VARCHAR.

  Pentru a include o instanță a caracterului ENCLOSED BY într-o valoare a unui câmp de intrare, aceasta trebuie fie dublată, fie precedată de caracterul ESCAPED BY. În caz contrar, va fi interpretat ca indicând sfârșitul câmpului. Pentru datele de ieșire, instanțele caracterului ENCLOSED BY din interiorul valorilor câmpurilor sunt precedate de caracterul ESCAPED BY.
- Caracterul ESCAPED BY este folosit pentru a modifica semnificația caracterelor speciale (escaping). În exemplele următoare, vom presupune că acest caracter escape este caracterul backslash (\). Pentru intrare, combinația \N (backslash-N) este interpretată ca NULL, iar \0 (backslash-ASCII 0) este interpretată ca un octet cu valoare zero. Pentru alte caractere, caracterul escape este eliminat și caracterul următor este folosit literal. De exemplu, \" este interpretat drept ghilimele duble, chiar dacă valorile câmpurilor sunt încadrate între ghilimele simple.

Pentru ieșire, caracterul escape este folosit pentru codificarea octeților de valori `NULL` și zero sub forma `\N`, respectiv `\0`. De asemenea, instanțele caracterelor `ESCAPED BY` și `ENCLOSED BY` sunt precedate de caracterul escape, așa cum sunt și primele caractere ale șirurilor de terminare ale câmpurilor și liniilor. În cazul în care caracterul `ESCAPED BY` este vid (`ESCAPED BY ''`), nu se va produce nici o modificare a semnificației caracterelor speciale. Pentru a specifica `\` drept caracter escape, dublați-l (`ESCAPED BY \\`).

Valoarea `LINES TERMINATED BY` specifică un caracter sau caractere care semnifică sfârșitul liniilor.

Dacă nu este dată nici una din clauzele `FIELDS` sau `LINES`, valorile prestabilite se prezintă ca și cum le-ați fi specificat astfel:

```
FIELDS
   TERMINATED BY '\t'
   ENCLOSED BY ''
   ESCAPED BY '\\'
LINES TERMINATED BY '\n'
```

Cu alte cuvinte, câmpurile din cadrul unei linii sunt delimitate prin tabulatori, fără a fi încadrate între ghilimele, backslash-ul este tratat drept caracter escape, iar liniile sunt terminate prin caractere linie nouă.

Dacă valorile `TERMINATED BY` și `ENCLOSED BY` pentru clauza `FIELDS` sunt ambele vide, se va folosi un format cu rânduri fixe, fără delimitatori între câmpuri. Valorile câmpurilor sunt citite (respectiv scrise, pentru ieșire) folosind lățimile de afișare ale coloanelor. De exemplu, coloanele `VARCHAR(15)` și `MEDIUMINT(5)` sunt citite sub formă de câmpuri de 15, respectiv 5 caractere pentru intrare. Pentru ieșire, coloanele sunt scrise folosind 15 caractere, respectiv 5 caractere. Valorile `NULL` sunt scrise sub formă de șiruri de spații.

Dacă este dată clauza `IGNORE n LINES`, primele `n` linii ale datelor de intrare sunt ignorate. De exemplu, dacă fișierul dumneavoastră de date are un rând cu antete de coloană pe care nu doriți să-l inserați în tabelul bazei de date, puteți folosi comanda `IGNORE 1 LINES`:

```
LOAD DATA LOCAL INFILE "tabelul_meu.txt"
   INTO TABLE tabelul_meu IGNORE 1 LINES
```

Dacă nu este specificată nici o listă de coloane, se va presupune că liniile de intrare conțin câte o valoare pentru fiecare coloană din tabel. Dacă o listă constă din una sau mai multe nume de coloane separate prin virgulă, liniile de intrare trebuie să conțină o valoare pentru fiecare coloană denumită. Coloanele care nu sunt specificate în listă sunt configurate la valorile lor prestabilite. Dacă dintr-o linie de intrare lipsește numărul de valori specificat, coloanele din care valorile lipsesc își vor primi valorile prestabilite.

Dacă aveți un fișier text delimitat prin tabulatori pe care l-ați creat în Windows, puteți folosi separatorul de coloane prestabilit, dar probabil că liniile se termină prin perechi retur de car-linie nouă. Pentru a încărca fișierul, specificați un alt caracter de terminare a liniei (`\r` indică un retur de car, iar `\n` indică un caracter linie nouă):

```
LOAD DATA LOCAL INFILE "tabelul_meu.txt"
   INTO TABLE tabelul_meu LINES TERMINATED BY "\r\n"
```


Din păcate, pentru fișierele create în Windows, vă puteți alege cu o înregistrare eronat în baza de date dacă programul care a creat fișierul de date folosește ciudata convenți din MS-DOS de inserție a caracterului Ctrl-Z la sfârșitul unui fișier, pentru a indic sfârșitul fișierului. Fie scrieți fișierul folosind un program care nu procedează astfel, fi ștergeți înregistrarea după încărcarea fișierului.

Fișierele cu valori separate prin virgulă (CSV) au virgule între câmpuri, iar câmpurile po fi încadrate între ghilimele duble. Presupunând că liniile se încheie prin caractere lini nouă, instrucțiunea `LOAD DATA` pentru încărcarea unui atare fișier se prezintă astfel:

```
LOAD DATA LOCAL INFILE "tabelul_meu.txt"
   INTO TABLE tabelul_meu LINES TERMINATED BY "," ENCLOSED BY "\""
```

Instrucțiunea următoare prezintă modul de citire a unui fișier pentru care câmpuril sunt separate prin caractere Ctrl-A (ASCII 1), iar liniile se încheie cu caractere Ctrl-I (ASCII 2):

```
LOAD DATA LOCAL INFILE "tabelul_meu.txt" INTO TABLE tabelul_meu
   FIELDS TERMINATED BY 0x01 LINES TERMINATED BY 0x02
```

LOCK TABLES

```
LOCK TABLES lista_blocare
```

Obține o blocare asupra tabelelor menționate în *`lista_blocare`*, așteptând, dacă est necesar, până la acumularea tuturor blocărilor. `LOCK TABLE` este sinonim cu `LOCK TABLES`

`lista_blocare` conține unul sau mai multe articole separate prin virgulă, unde fiecar articol respectă următoarea sintaxă:

```
nume_tabel [AS nume_alias] {READ | [LOW_PRIORITY] WRITE}
```

`READ` indică o blocare pentru citire. Aceasta îi blochează pe alți clienți care doresc să scri în tabel, dar le permite să citească tabelul. `WRITE` indică o blocare la scriere. Aceasta î blochează pe toți ceilalți clienți, indiferent dacă aceștia doresc să citească sau să scrie în tabel. O cerere de blocare `LOW_PRIORITY WRITE` permite altor programe de citire să citească tabelul, în cazul în care cererea așteaptă un alt client, care deja citește tabelul. Blocarea este obținută doar atunci când nu mai există programe care citesc tabelul.

`LOCK TABLES` permite specificarea unui alias, astfel încât puteți bloca un tabel la care veți face referire folosind aliasul respectiv într-o interogare ulterioară. Dacă folosiți de mai multe ori un tabel într-o interogare, trebuie să obțineți o blocare pentru fiecare instanță a tabelului, blocând aliasurile după necesități.

`LOCK TABLES` anulează toate blocările existente pe care le dețineți la un moment dat. Astfel, pentru a bloca mai multe tabele, trebuie să le blocați pe toate folosind o singură instrucțiune `LOCK TABLES`. Toate blocările deținute de un client atunci când acesta își termină execuția vor fi anulate automat.

Blocările de tip `LOW_PRIORITY` au fost introduse în versiunea MySQL 3.22.5.

```
LOCK TABLES elev READ, puncte WRITE, eveniment READ
LOCK TABLE membru READ
```

OPTIMIZE TABLE

```
OPTIMIZE TABLE nume_tabel
```

Recuperează spațiul irosit într-un tabel. Instrucțiunile DELETE, REPLACE și UPDATE pot avea ca rezultat apariția unor regiuni de spațiu nefolosit într-un tabel, mai ales în tabelele cu rânduri de lungime variabilă.

Instrucțiunea OPTIMIZE TABLE elimină spațiul irosit, reducând dimensiunea tabelului. Această instrucțiune a fost introdusă în MySQL 3.22.7. Alte opțiuni de întreținere a tabelului sunt programate a fi introduse în seria de versiuni MySQL 3.23, deci examinați manualul de referință MySQL pe suport electronic pentru a descoperi ultimele adăugiri.

REPLACE

```
REPLACE [LOW_PRIORITY | DELAYED] [INTO]
    nume_tabel [(lista_coloane)] VALUES (expr [, expr] ...) [, (...)] ...
REPLACE [LOW_PRIORITY | DELAYED] [INTO]
    nume_tabel [(lista_coloane)] SELECT ...
REPLACE [LOW_PRIORITY | DELAYED] [INTO]
    nume_tabel SET nume_coloana=expr [,nume_coloana=expr] ...
```

Instrucțiunea REPLACE este similară cu INSERT, cu deosebirea că, dacă un rând care trebuie inserat are o valoare a unui index UNIQUE care reproduce valoarea dintr-un rând deja existent în tabel, vechea valoare este ștearsă anterior introducerii noii valori. Din acest motiv, nu există nici o clauză IGNORE în sintaxa instrucțiunii REPLACE.

Pentru mai multe informații, vezi prezentarea instrucțiunii INSERT.

REVOKE

```
GRANT tip_priv [(lista_coloane)][, tip_priv [(lista_coloane)] ...]
    ON {*.* | * | nume_db.* | nume_db.nume_tabel | nume_tabel}
    FROM utilizator [, utilizator ] ...
```

Această instrucțiune revocă privilegiile utilizatorului sau utilizatorilor specificați. Clauzele *tip_priv*, *lista_coloane* și *utilizator* sunt specificate în același mod ca în instrucțiunea GRANT. Aceleași categorii de specificatori sunt permise și pentru clauza ON.

Instrucțiunea REVOKE nu elimină utilizatorul din tabelul de acordare user. Aceasta înseamnă că utilizatorul poate în continuare să se conecteze la serverul MySQL. Pentru a elimina în totalitate utilizatorul, trebuie să ștergeți manual intrările aferente acestuia din tabelele de acordare.

Instrucțiunea REVOKE a fost introdusă în versiunea MySQL 3.22.11.

Revocați toate privilegiile utilizatorului mare_si_tare@localhost:

```
REVOKE ALL ON *.* mare_si_tare@localhost
```

Revocați privilegiile care permit utilizatorului member_mgr să modifice tabelul membru din baza de date samp_db:

```
REVOKE INSERT,DELETE,UPDATE ON samp_db.membru
    FROM member_mgr@boa.snake.net
```


Revocați toate privilegiile pentru un singur tabel din baza de date menagerie aferente utilizatorului anonim care se conectează de la gazda locală:

```
REVOKE ALL ON menagerie.pet FROM ""@localhost
```

SELECT

```
SELECT
    [optiuni_selectare]
    lista_selectare
    [INTO OUTFILE 'nume_fisier' optiuni_export]
    [FROM lista_tabele
    [WHERE expresie_unde]
    [GROUP BY lista_coloane]
    [HAVING expresie_unde]
    [ORDER BY {unsigned_integer | nume_coloana | formula}[ASC | DESC] , ...]
    [LIMIT [offset,] randuri]
    [PROCEDURE nume_procedura ]
```

Regăsește rânduri din tabelele specificate. De fapt, toate clauzele sunt opționale, în afară de cuvântul cheie SELECT și de clauza *lista_selectare*, care acceptă instrucțiuni ce pur și simplu evaluează expresii:

```
SELECT "unu plus unu =", 1+1
```

Clauza optiuni_selectare, dacă există, poate conține următoarele opțiuni:

- ALL
  DISTINCT
  DISTINCTROW

  Aceste cuvinte cheie controlează dacă vor fi returnate sau nu rânduri duplicate. ALL determină returnarea tuturor rândurilor. Aceasta este opțiunea prestabilită. DISTINCT și DISTINCTROW specifică eliminarea rândurilor duplicate din setul de rezultate.

- HIGH_PRIORITY

  Prin specificarea acestei opțiuni, instrucțiunea dobândește o prioritate mai ridicată, dacă în mod normal ar fi trebuit să aștepte. Dacă alte instrucțiuni, precum INSERT și UPDATE, așteaptă să scrie în tabele specificate în instrucțiunea SELECT deoarece un alt client citește tabelele, opțiunea HIGH_PRIORITY face astfel încât instrucțiunii SELECT să i se acorde prioritate față de aceste instrucțiuni de scriere. Această opțiune trebuie folosită numai pentru instrucțiuni SELECT care se execută rapid și trebuie executată imediat, deoarece reduce viteza de execuție a instrucțiunilor de scriere. Opțiunea HIGH_PRIORITY a fost introdusă în MySQL 3.22.9.

- SQL_BIG_RESULT
  SQL_SMALL_RESULT

  Aceste cuvinte cheie arată că setul de rezultate va fi mare sau mic, informații pe care utilitarul de optimizare le poate folosi pentru a prelucra interogarea într-o manieră mai eficientă. Opțiunea SQL_SMALL_RESULT a fost introdusă în versiunea MySQL 3.22.12. Opțiunea SQL_BIG_RESULT a fost introdusă în versiunea MySQL 3.23.

- `STRAIGHT_JOIN`

  Forțează executarea unirii tabelelor în ordinea specificată în clauza `FROM`. Se poate folosi dacă credeți că utilitarul de optimizare nu face alegerea cea mai bună.

Clauza *`lista_selectare`* precizează coloanele de ieșire care trebuie returnate. Numele coloanelor, dacă sunt mai multe, trebuie separate prin virgule. Coloanele pot fi referințe la coloane din tabel sau expresii. Orice coloană poate primi un alias folosind sintaxa `AS` *`nume_alias`*. Apoi, aliasul devine numele coloanei în datele de ieșire și se poate face referire la acesta în clauzele `GROUP BY`, `ORDER BY` și `HAVING`. Totuși, nu puteți face referire la un alias într-o clauză `WHERE`.

Notația specială `*` are semnificația „toate coloanele din tabelele denumite în clauza `FROM`", iar *`nume_tabel`*`.*` este echivalentă cu „toate coloanele din tabelul denumit".

Rezultatul unei instrucțiuni `SELECT` poate fi scris într-un fișier *`nume_fisier`* folosind o clauză `INTO` '*`nume_fisier`*' `OUTFILE`. Trebuie să dispuneți de privilegiul `FILE`, iar fișierul nu trebuie să existe. Numele fișierului este interpretat folosind aceleași reguli care se aplică la citirea fișierelor non-`LOCAL` în instrucțiunea `LOAD DATA`. Sintaxa clauzei *`optiuni_export`* este aceeași ca a clauzei `optiuni_import` din instrucțiunea `LOAD DATA`. Pentru mai multe informații, consultați prezentarea instrucțiunii `LOAD DATA`.

Clauza `FROM` denumește unul sau mai multe tabele din care trebuie selectate rândurile. MySQL acceptă utilizarea următoarelor tipuri `JOIN` în instrucțiunile `SELECT`:

```
nume_tabele
    lista_tabele, nume_tabel
    lista_tabele [CROSS] JOIN nume_tabel
    lista_tabele INNER JOIN nume_tabel
    lista_tabele STRAIGHT_JOIN nume_tabel
    lista_tabele LEFT [OUTER] JOIN nume_tabel ON expr_conditionala
    lista_tabele LEFT [OUTER] JOIN nume_tabel USING (lista_coloane)
    lista_tabele NATURAL LEFT [OUTER] JOIN nume_tabel
    { oj lista_tabele LEFT OUTER JOIN nume_tabel ON expr_conditionala }
```

Tipurile de uniri selectează rânduri din tabelele denumite așa cum se va vedea în descrierile de mai jos, deși rândurile efectiv returnate clientului pot fi limitate de clauzele `WHERE`, `HAVING` sau `LIMIT`.

Pentru un singur tabel denumit, instrucțiunea `SELECT` regăsește rânduri din tabel.

Dacă sunt specificate mai multe tabele separate prin virgulă, instrucțiunea `SELECT` returnează toate combinațiile posibile de rânduri din tabele. Utilizarea clauzelor `JOIN`, `CROSS JOIN` sau `INNER JOIN` este similară cu utilizarea virgulelor. Clauza `STRAIGHT_JOIN` este similară, dar forțează utilitarul de optimizare să unească tabelele în ordinea în care sunt specificate. Această clauză se poate folosi dacă credeți că utilitarul de optimizare nu face alegerea cea mai bună.

Clauza `LEFT JOIN` regăsește rânduri din tabelele unite, dar forțează generarea unui rând pentru fiecare rând din tabelul din stânga, chiar dacă nu există nici un rând echivalent în tabelul din dreapta. Când nu există nici o corespondență, coloanele din tabelul din dreapta sunt returnate sub formă de valori `NULL`. Rândurile echivalente sunt determinate conform condiției specificate în clauza `ON expr_conditionala` sau în clauza `USING (lista_coloane)`. `expr_conditionala` este o expresie de forma care poate fi folosită în clauza `WHERE`. `lista_coloane` constă din una sau mai multe nume de coloane separate prin virgulă, iar toate aceste coloane trebuie să se regăsească în ambele tabele unite. `LEFT OUTER JOIN` este echivalentă cu `LEFT JOIN`, așa cum este și sintaxa care începe cu `oj` (inclusă pentru compatibilitatea cu ODBC).

`NATURAL LEFT JOIN` este echivalentă cu `LEFT JOIN USING (`*`lista_coloane`*`)`, unde *`lista_coloane`* denumește fiecare coloană comună pentru ambele tabele.

Tabelele pot primi aliasuri în clauza `FROM`, folosindu-se sintaxa *`nume_tabel nume_alias`* sau *`nume_tabel`* `AS` *`nume_alias`*. Referințele la o coloană cu alias inserate în altă parte a instrucțiunii `SELECT` trebuie să folosească aliasul respectiv.

Clauza `WHERE` specifică o expresie aplicată rândurilor selectate din tabelele denumite în clauza `FROM`. Rândurile care nu satisfac criteriile date de expresie sunt eliminate. Setul de rezultate poate fi restrâns și mai mult, prin intermediul clauzelor `HAVING` și `LIMIT`.

Clauza `GROUP BY lista_coloane` grupează rânduri din setul de rezultate conform coloanelor denumite în listă. Această clauză este folosită la specificarea unor funcții de sumar precum `COUNT()` sau `MAX()` în clauza `lista_selectie`. În lista `lista_selectie` se poate face referire folosind nume de coloane sau aliasuri, respectiv poziția acestora în lista de selecție. Pozițiile coloanelor sunt numerotate începând de la 1.

Clauza `HAVING` specifică o expresie secundară, folosită pentru a limita rândurile după ce acestea au satisfăcut condițiile denumite de clauza `WHERE`. Rândurile care nu satisfac condiția `HAVING` sunt respinse. Clauza `HAVING` este utilă pentru expresii care implică funcții de sumar, ce nu pot fi testate în clauza `WHERE`. Totuși, dacă o condiție este admisă în clauza `WHERE` sau în clauza `HAVING`, este preferabil ca aceasta să fie plasată în clauza `WHERE`, unde va fi supusă analizei utilitarului de optimizare.

Clauza `ORDER BY` indică modul de sortare a setului de rezultate. Ca și la `GROUP BY`, coloanele pot fi desemnate prin nume sau prin alias, respectiv prin poziția în cadrul listei de selecție a coloanelor. În mod prestabilit, coloanele de ieșire sunt sortate în ordine ascendentă. Pentru a specifica în mod explicit o ordine de sortare pentru o coloană, inserați expresia `ASC` (ascending – crescător) sau `DESC` (descending – descrescător). Începând de la versiunea MySQL 3.23.2, mai puteți specifica și o expresie drept coloană de sortare. De exemplu, `ORDER BY RAND()` returnează rândurile în ordine aleatoare.

Clauza `LIMIT` se poate folosi pentru a selecta un număr de rânduri din setul de rezultate. `LIMIT n` returnează primele `n` rânduri. `LIMIT m,n` returnează `n` rânduri, începând de la rândul `m`. Pentru `LIMIT`, rândurile sunt numerotate începând de la 0, nu de la 1.

`PROCEDURE` denumește o procedură căreia îi vor fi trimise datele din setul de rezultate înainte de a i se returna clientului un set de rezultate. Începând de la versiunea MySQL 3.23, puteți folosi `PROCEDURE ANALYSE()` pentru a obține informații despre caracteristicile datelor din coloanele denumite în lista de selecție a coloanelor.

Următoarele instrucțiuni prezintă unele moduri de utilizare a instrucțiunii `SELECT`. Pentru mai multe exemple, consultați capitolul 1, „Introducere în MySQL și SQL".

Selectează întregul conținut al unui tabel:

```
SELECT * FROM presedinte
```

Selectează întregul conținut, dar sortează după nume:

```
SELECT * FROM presedinte ORDER BY nume, prenume
```

Selectează înregistrările președinților născuți după data de "1900-01-01":

```
SELECT * FROM presedinte WHERE nastere = "1900-01-01"
```

Aceeași operație, dar se execută sortarea după data nașterii:

```
SELECT * FROM presedinte WHERE nastere = "1900-01-01" ORDER BY nastere
```

Determină care sunt statele reprezentate prin rânduri în tabelul membru:

```
SELECT DISTINCT stat FROM membru;
```

Selectează rânduri din tabelul membru și scrie coloanele sub formă de valori separate prin virgulă într-un fișier:

```
SELECT * INTO OUTFILE "/tmp/membru.txt"
  FIELDS TERMINATED BY "," FROM membru
```

Selectează primele cinci punctaje obținute la un anumit eveniment:

```
SELECT * FROM puncte WHERE eveniment_id = 9 ORDER BY puncte DESC LIMIT 5
```

SET

```
SET [OPTION] stabilire_optiuni, ...
```

Instrucțiunea SET este folosită pentru a specifica o varietate de opțiuni. Cel mai indicat este să omiteți cuvântul OPTION, deoarece acesta va fi eliminat într-o versiune ulterioară a sistemului MySQL.

Valorile stabilire_optiuni pot fi oricare dintre articolele prezentate în lista următoare:

- CHARACTER SET {*nume_setcaractere* | DEFAULT}

  Specifică setul de caractere folosit de către client. Șirurile trimise clientului, respectiv cele primite de la client, sunt mapate folosind acest set de caractere. Singurul set de caractere disponibil în prezent este cp1251_koi18. Numele setului de caractere DEFAULT restabilește setul de caractere prestabilit:

  ```
  SET CHARACTER SET cp1251_koi18
  SET CHARACTER SET DEFAULT
  ```

- INSERT_ID = *n*

  Specifică valoarea care va fi folosită în următoarea instrucțiune INSERT la inserția unei coloane AUTO_INCREMENT. Aceasta este folosită pentru prelucrarea jurnalului de actualizare.

  ```
  SET INSERT_ID = 1973
  ```

- LAST_INSERT_ID = *n*

  Specifică valoarea care va fi returnată de LAST_INSERT_ID(). Aceasta este folosită pentru prelucrarea jurnalului de actualizare.

  ```
  SET LAST_INSERT_ID = 48731
  ```

- PASSWORD [FOR *utilizator*] = PASSWORD("*parola*")

  În lipsa clauzei FOR, stabilește parola utilizatorului curent ca fiind "*parola*". Dacă această clauză există, stabilește parola pentru utilizatorul dat. Pentru a putea configura parola unui alt utilizator, trebuie să aveți privilegii pentru modificarea bazei de date mysql. *utilizator* este specificat în forma *nume_utilizator@nume_gazda*, folosind aceleași tipuri de valori pentru *nume_utilizator* și *nume_gazda* acceptabile pentru instrucțiunea GRANT.

  ```
  SET PASSWORD = PASSWORD("secret")
  SET PASSWORD FOR paul = PASSWORD("secret")
  SET PASSWORD FOR paul@localhost = PASSWORD("secret")
  SET PASSWORD FOR bill@%.bigcorp.com = PASSWORD("old-sneep")
  ```

- SQL_AUTO_IS_NULL = {0 | 1}

  Dacă această opțiune are valoarea 1, ultimul rând conținând o valoare AUTO_INCREMENT care a fost inserat poate fi selectat folosind o clauză WHERE de forma WHERE *auto_inc_col* IS NULL. Această clauză este folosită de unele programe ODBC, precum Access. Valoarea prestabilită este 1. Această opțiune a fost introdusă în MySQL 3.23.5.

  ```
  SET SQL_AUTO_IS_NULL = 0
  ```

- SQL_BIG_SELECTS = {0 | 1}

  Instrucțiunile SELECT care vor returna mai mult de max_join_size rânduri sunt permise dacă această opțiune are valoarea 1. În caz contrar, interogarea va eșua. Valoarea prestabilită este 1.

  ```
  SET SQL_BIG_SELECTS = 0
  ```

- SQL_BIG_TABLES = {0 | 1}

  Dacă această opțiune are valoarea 1, toate tabelele temporare sunt stocate pe disc, nu în memorie. Performanțele sunt mai lente, dar instrucțiunile SELECT care necesită tabele temporare mari nu vor genera erori de tip „tabele pline". Valoarea prestabilită este 0 (stocarea tabelelor temporare în memorie). În mod normal, această opțiune nu este necesară pentru MySQL 3.23 și versiunile ulterioare.

  ```
  SET SQL_BIG_TABLES = 1
  ```

- SQL_LOG_OFF = {0 | 1}

  Dacă această opțiune are valoarea 1, interogările clientului curent nu sunt consemnate în fișierul jurnal general. Dacă opțiunea are valoarea 0, este activată consemnarea în jurnal pentru clientul respectiv. Clientul trebuie să aibă privilegiul PROCESS pentru ca această instrucțiune să aibă efect. Procesul de consemnare în jurnalul de actualizare nu este afectat.

  ```
  SET SQL_LOG_OFF = 1
  ```

- SQL_LOG_UPDATE = {0 | 1}

  Este similară cu instrucțiunea SQL_LOG_OFF (inclusiv necesitatea privilegiului PROCESS), cu excepția faptului că afectează consemnarea în jurnalul de actualizare și nu în jurnalul general. SQL_LOG_UPDATE a fost introdusă în MySQL versiunea 3.22.5.

  ```
  SET SQL_LOG_UPDATE = 1
  ```

- SQL_LOW_PRIORITY_UPDATES = {0 | 1}

  Dacă această opțiune are valoarea 1, instrucțiunile care aduc modificări conținutului unui tabel (DELETE, INSERT, REPLACE, UPDATE) așteaptă până când nici o instrucțiune SELECT pentru tabelul respectiv nu mai este activă sau suspendată. Instrucțiunile

SELECT care sosesc în timp ce o altă instrucțiune este activă încep să se execute imediat, în loc să aștepte instrucțiunile de modificare cu prioritate redusă.

Opțiunea SQL_LOW_PRIORITY_UPDATES a fost introdusă în MySQL 3.22.5.

```
SET SQL_LOW_PRIORITY_UPDATES = 0
```

- SQL_SELECT_LIMIT = {*n* | DEFAULT}

Specifică numărul maxim de înregistrări ce vor fi returnate de către o instrucțiune SELECT. Prezența unei clauze LIMIT explicite într-o instrucțiune are prioritate în raport cu această opțiune. Valoarea prestabilită este „fără limite". Valoarea DEFAULT restaurează valoarea prestabilită, dacă ați modificat-o.

```
SET SQL_SELECT_LIMIT = 100000
SET SQL_SELECT_LIMIT = DEFAULT
```

- SQL_WARNINGS = {0 | 1}

Dacă această opțiune este configurată la valoarea 1, MySQL raportează numărul de avertizări chiar și pentru inserția unui singur rând. În mod normal, numerele de avertizări sunt raportate numai pentru instrucțiunile INSERT care inserează mai multe rânduri. Valoarea prestabilită este 0. Opțiunea SQL_WARNINGS a fost introdusă în versiunea MySQL 3.22.11.

```
SET SQL_WARNINGS = 1
```

- TIMESTAMP = {*valoare_amprenta* | DEFAULT}

Specifică o valoare TIMESTAMP. Este folosită pentru prelucrarea jurnalului de actualizare.

```
SET TIMESTAMP = DEFAULT
```

SHOW

```
SHOW COLUMNS FROM nume_tabel [FROM nume_db] [LIKE model]
SHOW DATABASES [LIKE model]
SHOW GRANTS FOR nume_utilizator
SHOW INDEX FROM nume_tabel [FROM nume_db]
SHOW PROCESSLIST
SHOW STATUS
SHOW TABLE STATUS [FROM nume_db][LIKE model]
SHOW TABLES [FROM nume_db] [LIKE model]
SHOW VARIABLES [LIKE model]
```

Diferitele forme ale instrucțiunii SHOW furnizează informații despre baze de date, tabele, coloane și indexuri, respectiv informații despre modul de operare a serverului. Numeroase forme acceptă o clauză opțională FROM *nume_db*, permițându-vă să specificați baza de date ale cărei informații trebuie prezentate. În cazul în care clauza nu este prezentă, se folosește baza de date prestabilită. Unele forme permit o clauză opțională LIKE *model* pentru limitarea ieșirii la valorile care corespund modelului. Modelul trebuie să fie un șir sau un model SQL care să conțină caracterele % sau _.

SHOW COLUMNS

Instrucțiunea SHOW COLUMNS enumeră coloanele pentru tabelul dat. SHOW FIELDS este sinonimă cu instrucțiunea SHOW COLUMNS.

```
SHOW COLUMNS FROM presedinte
SHOW FIELDS FROM presedinte
SHOW COLUMNS FROM presedinte FROM samp_db
SHOW COLUMNS FROM tables_priv FROM mysql LIKE "%priv".
```

Datele de ieșire ale instrucțiunii SHOW COLUMNS conțin următoarele coloane:

- Field

Numele coloanei din tabel.

- Type

Tipul coloanei.

- Null

Dacă coloana poate sau nu să conțină valori NULL. Poate fi YES sau blanc (nu).

- Key

Dacă coloana este indexată sau nu.

- Default

Valoarea prestabilită a coloanei.

- Extra

Informații suplimentare despre coloană.

- Privileges

Privilegiile aferente coloanei. Aceste informații sunt prezentate numai începând de la versiunea MySQL 3.23.

SHOW DATABASES

Instrucțiunea SHOW DATABASES enumeră bazele de date disponibile pe gazda serverului.

```
SHOW DATABASES
SHOW DATABASES LIKE "test%"
```

SHOW GRANTS

Instrucțiunea SHOW GRANTS afișează informațiile de acordare cu privire la utilizatorul specificat, care trebuie specificate sub forma *nume_utilizator@nume_gazda*, folosind aceleași tipuri de valori pentru *nume_utilizator* și *nume_gazda* care sunt acceptabile pentru instrucțiunea GRANT. Instrucțiunea SHOW GRANTS a fost introdusă în MySQL versiunea 3.23.4.

```
SHOW GRANTS FOR root@localhost
SHOW GRANTS FOR ""@pit-viper.snake.net
```

SHOW INDEX

Instrucțiunea SHOW INDEX afișează informații despre indexurile unui tabel. Instrucțiunea SHOW KEYS este sinonimă cu SHOW INDEX.

```
SHOW INDEX FROM puncte
SHOW KEYS FROM puncte
SHOW INDEX FROM samp_db.puncte
SHOW INDEX FROM puncte FROM samp_db
```

Datele de ieșire ale funcției SHOW INDEX conțin următoarele coloane:

- Table

  Numele tabelului care conține indexul.
- Non_unique

  Această valoare este 1 dacă indexul poate conține valori duplicate, respectiv 0 dacă indexul nu poate conține asemenea valori.
- Key_name

  Numele indexului.
- Seq_in_index

  Numărul coloanei din index. Coloanele sunt numerotate începând de la 1.
- Column_name

  Numele coloanei.
- Collation

  Ordinea de sortare a coloanelor în cadrul indexului. Valorile pot fi A (ascending - crescător), D (descending - descrescător) sau NULL (fără sortare). Cheile descrescătoare nu sunt încă disponibile, dar vor fi implementate în viitor.
- Cardinality

  Numărul de valori unice din index. Această valoare este actualizată prin rularea programelor myisamchk sau isamchk cu opțiunea --analyze.
- Sub_part

  Lungimea prefixului, dacă este indexat numai un prefix al coloanei. Are valoarea NULL dacă este indexată întreaga coloană.
- Packed

  Modul în care este împachetată cheia, respectiv NULL dacă nu este împachetată. Această coloană este prezentată numai începând de la versiunea 3.23.
- Comment

  Rezervat pentru comentarii interne referitoare la index. Această coloană este prezentată numai pentru MySQL 3.23 și versiunile ulterioare.

SHOW PROCESSLIST

Funcția SHOW PROCESSLIST afișează informații despre firele care se execută în cadrul serverului. Datele de ieșire conțin următoarele coloane:

- Id

  Numărul de identificare a firului pentru client.
- User

  Numele de client asociat cu firul.

- Host

  Gazda de la care este conectat clientul.
- db

  Baza de date prestabilită pentru fir.
- Command

  Comanda executată de fir.
- Time

  Intervalul de timp folosit de comanda aflată în execuție în fir, exprimat în secunde.
- State

  Informații despre acțiunile sistemului MySQL în timpul prelucrării unei instrucțiun SQL. Această valoare poate fi utilă pentru semnalarea unei probleme a sistemulu MySQL sau la inserția în lista de corespondență pentru MySQL a unei întrebări privinc motivul pentru care firul rămâne într-o anumită stare pentru o perioadă de timp ma îndelungată.
- Info

  Interogarea care se execută.

SHOW STATUS

Instrucțiunea SHOW STATUS afișează variabilele de stare ale serverului și valorile acestora

- Aborted_clients

  Numărul de conexiuni client întrerupte datorită clienților care nu au închis conexiunea în mod adecvat.
- Aborted_connects

  Numărul de încercări ratate de conectare la server.
- Connections

  Numărul de încercări de conectare la server (atât reușite, cât și nereușite). Dacă această opțiune are o valoare foarte mare, puteți folosi conexiuni persistente pentru clienții dumneavoastră, dacă este posibil.
- Created_tmp_tables

  Numărul tabelelor temporare create în timpul prelucrării interogărilor.
- Delayed_errors

  Numărul de erori care s-au produs în timpul prelucrării rândurilor INSERT DELAYED.
- Delayed_insert_threads

  Numărul curent de rutine de tratare INSERT_DELAYED.
- Delayed_writes

  Numărul de rânduri INSERT DELAYED care au fost scrise.
- Flush_commands

  Numărul de comenzi FLUSH care au fost executate.

- `Handler_delete`

  Numărul de cereri de ștergere a unui rând dintr-un tabel.
- `Handler_read_first`

  Numărul de cereri de citire a primului rând dintr-un tabel.
- `Handler_read_key`

  Numărul de cereri de citire a unui rând în funcție de valoarea dintr-un index.
- `Handler_read_next`

  Numărul de cereri de citire a rândului următor din ordinea de indexare.
- `Handler_read_rnd`

  Numărul de cereri de citire a unui rând în funcție de poziția rândului. Dacă acest număr este cu mult mai mare decât toate celelalte numere raportate, fie executați multe interogări care impun baleierea integrală a tabelului, fie nu folosiți indexurile în mod adecvat.
- `Handler_update`

  Numărul de cereri de actualizare a unui rând dintr-un tabel.
- `Handler_write`

  Numărul de cereri de inserție a unui rând într-un tabel.
- `Key_blocks_used`

  Numărul de blocuri care vor fi utilizate în zona cache a indexului.
- `Key_read_requests`

  Numărul de cereri de citire a unui bloc din zona cache a indexului.
- `Key_reads`

  Numărul de citiri fizice de blocuri de index de pe disc.
- `Key_write_requests`

  Numărul de cereri de scriere a unui bloc în zona cache a indexului.
- `Key_writes`

  Numărul de scrieri fizice de blocuri de index pe disc.
- `Max_used_connections`

  Numărul maxim de conexiuni care au fost deschise simultan.
- `Not_flushed_delayed_rows`

  Numărul rândurilor care așteaptă să fie scrise pentru interogările `INSERT DELAYED`.
- `Not_flushed_key_blocks`

  Numărul blocurilor din zona cache a cheilor care au fost modificate, dar care nu au fost încă evacuate pe disc.
- `Open_files`

  Numărul de fișiere deschise.

- `Open_streams`

  Numărul fluxurilor deschise. Un flux este un fișier deschis cu `fopen()`; acest proced se aplică numai fișierelor jurnal.
- `Open_tables`

  Numărul tabelelor deschise.
- `Opened_tables`

  Numărul total al tabelelor care au fost deschise. Dacă acest număr are o valoare mar se recomandă mărirea dimensiunii zonei cache a tabelului.
- `Questions`

  Numărul interogărilor recepționate de server (include atât interogările reușite, cât cele nereușite).
- `Slow_queries`

  Numărul interogărilor care necesită un timp de execuție mai lung dec `long_query_time`.
- `Threads_connected`

  Numărul de conexiuni deschise la un moment dat.
- `Threads_running`

  Numărul firelor care nu sunt inactive.
- `Uptime`

  Intervalul de timp (în secunde) scurs de la momentul pornirii serverului.

SHOW TABLE STATUS

Instrucțiunea `SHOW TABLE STATUS` afișează informații cu caracter descriptiv cu privire la ta belele dintr-o bază de date. `SHOW TABLE STATUS` a fost introdusă în versiunea MySQL 3.2

```
SHOW TABLE STATUS
SHOW TABLE STATUS FROM samp_db
SHOW TABLE STATUS FROM mysql LIKE "%priv"
```

Datele de ieșire ale instrucțiunii `SHOW TABLE STATUS` conțin următoarele coloane:

- `Name`

  Numele tabelului.
- `Type`

  Tipul tabelului; acesta poate fi NISAM (ISAM), MyISAM sau HEAP.
- `Row_format`

  Formatul de stocare a rândurilor; acesta poate fi `Fixed` (rânduri de lungime fixă) `Dynamic` (rânduri de lungime variabilă) sau `Compressed`.
- `Rows`

  Numărul rândurilor dintr-un tabel.
- `Avg_row_length`

  Numărul mediu de octeți folosit de rândurile tabelului.

- `Data_length`
  Dimensiunea efectivă (în octeți) a fișierului de date al tabelului.
- `Max_data_length`
  Dimensiunea maximă (în octeți) la care poate ajunge fișierul de date al tabelului.
- `Index_length`
  Dimensiunea efectivă (în octeți) a fișierului index.
- `Data_free`
  Numărul de octeți nefolosiți din fișierul de date. Dacă acest număr are o valoare foarte mare, se recomandă emiterea unei instrucțiuni `OPTIMIZE TABLE` pentru tabel.
- `Auto_increment`
  Următoarea valoare care va fi generată pentru o coloană `AUTO_INCREMENT`.
- `Create_time`
  Momentul creării tabelului.
- `Update_time`
  Momentul celei mai recente modificări a tabelului.
- `Check_time`
  Momentul ultimei verificări sau reparări a tabelului cu ajutorul utilitarului `myisamchk`; valoarea este `NULL` dacă tabelul nu a fost niciodată verificat sau reparat.
- `Create_options`
  Opțiunile suplimentare care au fost specificate în clauza `optiuni_tabel` a instrucțiunii `CREATE TABLE` care a creat tabelul.
- `Comment`
  Textul oricărui comentariu specificat la crearea tabelului.

SHOW TABLES

Instrucțiunea `SHOW TABLES` afișează o listă a tabelelor dintr-o bază de date.

```
SHOW TABLES
SHOW TABLES FROM samp_db
SHOW TABLES FROM mysql LIKE "%priv"
```

SHOW VARIABLES

Instrucțiunea `SHOW VARIABLES` afișează o listă a variabilelor serverului și a valorilor acestora. Variabilele serverului sunt descrise în prezentarea programului `mysqld` din Anexa E, „Referință de programe MySQL".

```
SHOW VARIABLES
SHOW VARIABLES LIKE "%thread"
```

UNLOCK TABLES

```
UNLOCK TABLES
```

Această instrucțiune anulează toate blocările de tabele deținute de clientul curent. Instrucțiunea `UNLOCK TABLE` este sinonimă cu `UNLOCK TABLES`.

UPDATE

```
UPDATE [LOW_PRIORITY] nume_tabel
    SET nume_coloana=expr [, nume_coloana=expr ] ...
    [WHERE unde_expr] [LIMIT n]
```

Aduce modificări conținutului rândurilor existente în tabelul `nume_tabel`. Rândurile d modificat sunt cele selectate prin expresia specificată în clauza `WHERE`. Pentru rândurile selec tate, fiecare coloană denumită în clauza `SET` primește valoarea expresiei corespunzătoare.

```
UPDATE membru SET expirare = NULL, telefon = "197-602-4832"
    WHERE membru_id = 14
```

Dacă nu este dată nici o clauză `WHERE`, *toate rândurile tabelului vor fi actualizate.*

Instrucțiunea `UPDATE` returnează numărul rândurilor care au fost actualizate. Cu toate acestea un rând nu este considerat ca fiind actualizat decât atunci când se modifică efectiv valoarea une coloane. A atribui unei coloane o valoare pe care aceasta o conține deja nu se consideră c afectează rândul. Dacă aplicația dumneavoastră *trebuie* să cunoască numărul rândurilor care a corespuns clauzei `WHERE` indiferent dacă instrucțiunea `UPDATE` a modificat vreo valoare, trebui să specificați indicatorul `CLIENT_FOUND_ROWS` atunci când stabiliți o conexiune cu serverul Consultați prezentarea funcției `mysql_real_connect()` în Anexa F, „Referință API C".

Clauza `LOW_PRIORITY` determină amânarea execuției instrucțiunii până când nici u client nu mai citește din tabel. Această clauză a fost introdusă în MySQL 3.22.5.

Dacă este dată clauza `LIMIT`, valoarea `n` specifică numărul maxim de rânduri care vor f actualizate. Clauza `LIMIT` a fost introdusă în MySQL 3.23.3.

USE

```
USE nume_db
```

Selectează *nume_db* pentru a o transforma în bază de date curentă (baza de date presta bilită pentru referirile la tabele care nu includ un nume explicit al unei baze de date) Instrucțiunea `USE` eșuează dacă baza de date nu există sau dacă nu aveți nici un privilegi pentru a obține accesul la aceasta.

Sintaxa comentariilor

Această secțiune descrie modul de scriere a comentariilor în programul dumneavoastr SQL. De asemenea, semnalează un neajuns al programului client `mysql` în raport c interpretarea comentariilor.

Comentariile sunt frecvent folosite în fișierele de interogare care sunt executate folosin `mysql` în modul batch, deci trebuie să fiți foarte atent la această limitare atunci cân scrieți asemenea fișiere.

Serverul MySQL înțelege trei tipuri de comentarii:

- Toate caracterele cuprinse între `#` și sfârșitul liniei sunt tratate drept comentariu Această sintaxă este similară celei folosite în majoritatea interpretoarelor și în Perl.
- Toate caracterele cuprinse între `/*` și `*/` sunt tratate drept comentariu. Această form de comentarii se poate extinde pe mai multe linii. Această sintaxă este similară cele folosite în limbajul C.

- Începând de la versiunea MySQL 3.23.3, un comentariu poate începe folosind caracterele „-- “ (două cratime și un spațiu); toate caracterele cuprinse între cratime și sfârșitul liniei sunt tratate drept comentariu. Acest stil de comentariu este folosit de alte sisteme de baze de date, cu excepția faptului că spațiul nu este obligatoriu. MySQL impune caracterul spațiu pentru eliminarea ambiguităților, astfel încât expresii precum *valoare1–valoare2*, unde *valoare2* este negativ, nu vor fi tratate drept comentarii.

La executarea interogărilor, serverul ignoră comentariile, cu excepția comentariilor de tip C care încep cu `/*!`, care se bucură de un tratament special. Textul comentariului trebuie să conțină cuvinte cheie din SQL, iar cuvintele cheie vor fi tratate de către serverul MySQL ca parte a instrucțiunii în care apare comentariul. De exemplu, următoarele linii sunt considerate de server ca fiind echivalente:

```
INSERT LOW_PRIORITY INTO tabelul_meu SET ...
INSERT /*! LOW_PRIORITY */ INTO tabelul_meu SET ...
```

Această formă de comentariu este destinată utilizării pentru extensii și cuvinte cheie specifice sistemului MySQL. MySQL va recunoaște cuvintele cheie, iar alte servere SQL le vor ignora. Astfel, este facilitată scrierea interogărilor care folosesc caracteristici specifice sistemului MySQL, dar care sunt funcționale și în alte sisteme de baze de date. Stilul de comentariu `/*!` a fost introdus în versiunea MySQL 3.22.7.

Începând de la versiunea MySQL 3.22.26, secvența `/*!` poate fi urmată de un număr de versiune, pentru a indica sistemului MySQL să ignore comentariul dacă numărul versiunii serverului este anterior versiunii respective. Comentariul din următoarea instrucțiune `UPDATE` este ignorat dacă versiunea serverului este anterioară versiunii 3.23.3:

```
UPDATE tabelul_meu SET coloana_mea = 100
    WHERE coloana_mea  100 /*!32303 LIMIT 100 */
```

Clientul `mysql` este mai limitat decât serverul MySQL în ceea ce privește capacitatea de a înțelege comentariile. `mysql` se încurcă dacă sunt inserate anumite construcții în comentariile scrise în stil C, deoarece folosește un analizor mai puțin sofisticat decât serverul. De exemplu, un caracter de citare care apare în interiorul unui comentariu scris în stil C va deruta programul, făcându-l să creadă că analizează un șir, și acesta va continua să caute sfârșitul șirului până la găsirea unui caracter de citare corespunzător. Instrucțiunile următoare ilustrează acest comportament:

```
mysql> SELECT /* I have no quote */ 1;
+---+
| 1 |
+---+
| 1 |
+---+
mysql> SELECT /* I've got a quote */ 1;
    '>
```

Clientul `mysql` analizează fără probleme prima instrucțiune, o trimite la server pentru execuție și afișează un alt prompt `mysql`. Cea de-a doua instrucțiune conține un comentariu cu un semn al citării fără pereche. Ca rezultat, `mysql` intră în modul de analiză a șirului și continuă să se găsească în modul respectiv și după ce ați introdus linia, așa cum se vede din promptul `'`. Pentru a ieși din această situație, tastați un caracter de citare corespunzător, urmat de o comandă `\c` pentru a anula interogarea. Pentru mai multe informații privind semnificația prompturilor clientului `mysql`, consultați Anexa E.

ANEXA E

Referință de programe MySQL

Această anexă descrie programele MySQL prezentate în lista următoare. Ulterior p
parcursul acestei anexe, fiecare program este descris mai detaliat, fiind dată inclusiv
scurtă descriere a rolului acestuia, a sintaxei de invocare, a opțiunilor pe care le accept
și o descriere a variabilelor sale interne.

- `isamchk` și `myisamschk`

  Utilitare pentru verificarea și repararea tabelelor, care execută analiza distribuți
  cheilor, precum și dezactivarea și reactivarea indexurilor.

- `myisampack` și `isampack`

  Utilitare pentru generarea de tabele comprimate numai pentru citire.

- `mysql`

  Program interactiv, care dispune de caracteristici de editare a liniilor pentru expediere
  interogărilor către serverul MySQL; poate fi de asemenea folosit în modul batch, pen
  tru executarea interogărilor stocate într-un fișier.

- `mysql.server`

  Script pentru pornirea și oprirea serverului MySQL.

- `mysqlaccess`

  Script pentru testarea privilegiilor de acces.

- `mysqladmin`

  Utilitar pentru efectuarea de operații administrative.

- `mysqlbug`

  Script pentru generarea rapoartelor de semnalare a hibelor.

- `mysqld`

  Serverul MySQL; acest program trebuie să ruleze pentru a permite clienților să aib
  acces la bazele de date administrate de server.

- `mysqldump`

  Utilitar pentru transferul conținutului tabelelor dintr-o bază de date.

- `mysqlimport`

  Utilitar pentru încărcarea unor mari cantități de date în tabele.

- `mysqlshow`

  Utilitar care furnizează informații despre baze de date sau tabele.

- `safe_mysqld`

  Script pentru pornirea și monitorizarea serverului MySQL.

În descrierea sintaxei, informațiile opționale sunt indicate prin paranteze drepte ([ ]).

Specificarea opțiunilor de program

Majoritatea programelor MySQL acceptă numeroase opțiuni care influențează modul de operare a programului. Opțiunile pot fi specificate în linia de comandă sau în fișierele cu opțiuni. În plus, unele opțiuni pot fi specificate prin configurarea variabilelor de mediu. Opțiunile specificate în linia de comandă au prioritate în raport cu opțiunile specificate în orice alt mod, iar opțiunile din fișierele cu opțiuni au prioritate în raport cu valorile variabilelor de mediu.

Obținerea de asistență on-line de la programele MySQL

Fiecare program MySQL, cu excepția utilitarului `mysqlbug` și a scripturilor `mysql.server` și `safe_mysqld`, înțelege opțiunea `--help`. Această opțiune constituie o modalitate rapidă de a obține asistență on-line cu privire la un program chiar din partea programului respectiv. De exemplu, dacă nu știți să folosiți `mysqlimport`, invocați-l în modul următor, pentru a obține instrucțiuni:

```
% mysqlimport --help
```

Opțiunea `-?` este aceeași ca `help`, deși shell-ul dumneavoastră poate interpreta caracterul `?` drept caracter de înlocuire pentru numele fișierelor:

```
% mysqlimport -?
mysqlimport: No match.
```

Dacă vi se întâmplă așa ceva, încercați următoarea comandă:

```
% mysqlimport -\?
```

Majoritatea opțiunilor au atât o formă scurtă (alcătuită dintr-o singură literă), cât și o formă lungă (cuvânt complet). Opțiunile `-?` și `--help` prezentate anterior reprezintă un exemplu în acest sens. Dacă o opțiune cu formă scurtă este urmată de o valoare, în majoritatea cazurilor opțiunea și valoarea pot fi separate prin spațiu. De exemplu, când specificați un nume de utilizator, `-upaul` este echivalent cu `-u paul`. Opțiunea `-p` (parola) este o excepție; orice valoare a parolei trebuie să urmeze imediat după litera `p`, fără nici un spațiu între ele.

Fiecare descriere de program prezintă toate opțiunile pe care un program le acceptă în acest moment. Dacă un program nu pare să recunoască o opțiune enumerată în descriere, s-ar putea să dispuneți de o versiune mai veche a programului, anterioară versiunii în care s-a adăugat opțiunea. (Verificați cu mare atenție sintaxa, pentru a vă asigura dacă nu cumva pur și simplu ați specificat incorect opțiunea respectivă.)

Opțiuni standard ale programelor MySQL

Numeroase opțiuni au o semnificație standard în mai multe programe MySQL. În loc de a le prezenta în descrierea fiecărui program care le acceptă, le vom enumera aici o singură dată, iar secțiunea „Opțiuni standard acceptate" din fiecare descriere de program

indică opțiunile pe care le acceptă fiecare program. (În secțiunea respectivă se vor prezenta numai numele în format lung, dar programele acceptă și opțiunile în format scurt corespunzătoare, cu excepția situațiilor când se specifică altceva.)

Opțiunile standard sunt:

- `-C, --compress`

  Permite utilizarea comprimării pentru protocolul folosit la comunicația între client și server, dacă serverul o acceptă. Această opțiune este folosită numai de programele client. A fost introdusă în MySQL 3.22.3.

- `-#` *`optiuni_depanare`*`, --debug=`*`optiuni_depanare`*

  Activează datele de ieșire pentru depanare. Această opțiune are efect numai dacă MySQL a fost construit având inclus suportul pentru depanare. Șirul *`optiuni_depanare`* constă din opțiuni separate printr-un caracter două puncte. O valoare caracteristică este `d:t:o,`*`nume_fisier`*, care permite depanarea, activează introducerea apelurilor de funcții și urmărirea ieșirilor, și trimite datele de ieșire în fișierul *`nume_fisier`*.

  Dacă vă așteptați la o activitate intensă de depanare, trebuie să examinați fișierul `dbug/dbug.c` din distribuția sursă MySQL, pentru o descriere a tuturor opțiunilor pe care le puteți folosi.

- `--defaults-file=`*`nume_cale`*

  Precizează calea de acces spre un fișier cu opțiuni. În mod normal, programele caută fișierele cu opțiuni în numeroase locații, dar, dacă este specificată opțiunea `--defaults-file`, atunci va fi citit numai fișierul denumit. Dacă folosiți această opțiune, trebuie ca opțiunea respectivă să fie menționată prima în linia de comandă. Opțiunea a fost introdusă în versiunea MySQL 3.22.23.

- `-?, --`

  Indică programului să afișeze un mesaj de asistență și să-și încheie execuția.

- `-h` *`nume_gazda`*`, --host=`*`nume_gazda`*

  Specifică gazda la care se va stabili conexiunea (adică gazda unde rulează serverul). Această opțiune este folosită numai de către programele client.

- `-no-defaults`

  Dezactivează utilizarea oricăror fișiere cu opțiuni. Dacă folosiți opțiunea respectivă aceasta trebuie să fie prima opțiune din linia de comandă.

- `-p[`*`parola`*`],-password[=`*`parola`*`]`

  Specifică parola care se va folosi pentru conectarea la server. Dacă `parola` nu este specificată după numele opțiunii, programul vă va solicita să introduceți o parolă. Dacă `parola` este specificată, trebuie să urmeze imediat după numele opțiunii, fără spațiu între ele. Această opțiune este folosită numai de către programele client.

- `-W, --pipe`

  Folosește un canal denumit pentru conectarea la server. Această opțiune este folosită numai pentru programe client care rulează sub Windows.

- `-P numar_port, --port=numar_port`

  Pentru programele client, acesta este numărul portului pe care îl vor folosi acestea pentru a se conecta la server. Acesta este portul folosit pentru conexiunile TCP/IP (conexiuni unde gazda nu este `localhost`). Pentru `mysqld`, această opțiune specifică portul unde serverul va aștepta conexiunile TCP/IP.

- `--print-defaults`

  Afișează valorile opțiunilor care vor fi folosite dacă invocați programul fără opțiuni în linia de comandă. Opțiunea prezintă valorile care vor fi selectate din fișierele cu opțiuni (și din variabilele de mediu). `--print-defaults` este utilă pentru verificarea configurării adecvate a unui fișier cu opțiuni. Puteți folosi `--print-defaults` pentru a determina dacă opțiunile sunt preluate dintr-un fișier cu opțiuni. Dacă folosiți această opțiune, trebuie să fie menționată prima în linia de comandă.

- `-O var=valoare, --set-variable var=valoare`

  Numeroase programe au variabile (parametri de operare) pe care le puteți configura. Opțiunea `--set-variable` vă permite să configurați aceste variabile. *`var`* este numele variabilei, iar *`valoare`* este valoarea care va fi atribuită variabilei. Valorile sunt exprimate în octeți, pentru variabilele care reprezintă lungimi sau dimensiuni ale bufferelor. Valorile pot fi specificate sub formă de număr exact, respectiv ca număr cu sufixul `K` sau `M`, pentru a simboliza kiloocteți sau megaocteți. Sufixele nu sunt sensibile la diferența între majuscule și minuscule: `k` și `m` sunt echivalente cu `K`, respectiv `M`.

  Variabilele fiecărui program sunt prezentate în descrierea programului din această anexă și sunt de asemenea afișate atunci când invocați programul cu opțiunea `--help`.

- `-s, --silent`

  Specifică modul silențios. Aceasta nu înseamnă în mod obligatoriu că programul este complet silențios, ci numai că furnizează mai puține date de ieșire ca de obicei. Mai multe programe permit specificarea acestei opțiuni de mai multe ori, pentru a determina programul să devină tot mai silențios (operația se poate executa și în fișierele cu opțiuni).

- `-S nume_cale, --socket=nume_cale`

  Pentru programele client, aceasta este numele căii de acces complete a fișierului soclu care se va folosi pentru conectarea la server. Pentru conexiunile cu `localhost` se va folosi un fișier soclu.

- `-u nume_utilizator --user=nume_utilizator`

  Pentru programe client, acesta este numele de utilizator MySQL care se va folosi pentru conectarea la server. Dacă această opțiune nu este specificată, valoarea prestabilită este numele dumneavoastră de conectare sub UNIX, respectiv `ODBC` sub Windows. Pentru `mysqld`, această opțiune indică numele contului UNIX care se va folosi pentru rularea serverului. (Serverul trebuie lansat sub contul `root` pentru ca această opțiune să fie utilizabilă.)

- `-v, --verbose`

  Specifică modul amănunțit; programul produce mai multe date de ieșire decât de obicei. Numeroase programe permit specificarea acestei opțiuni de mai multe ori, pentru a determina programul să devină tot mai „vorbăreț" (operația se poate executa și în fișierele cu opțiuni).

- `-V, --version`

  Indică programului să afișeze șirul cu informații despre versiune și să-și încheie execuția.

Fișiere cu opțiuni

Fișierele cu opțiuni au fost introduse în MySQL 3.22. Acestea vă permit să stocați opțiuni de program MySQL, astfel încât să nu fie necesar să le tastați în linia de comandă de fiecare dată când invocați un program. Fișierele cu opțiuni sunt citite de către programele `isamchk`, `myisamchk`, `myisampack`, `mysql`, `mysqladmin`, `mysqld`, `mysqldump`, `mysqlimport`, `mysql.server` și `pack_isam`. Puteți găsi un exemplu de fișier cu opțiuni, denumit `my-example.cnf`, în catalogul `share/mysql` din catalogul de instalare MySQL sau în catalogul `support-files` al unei distribuții sursă.

Orice opțiune specificată într-un fișier cu opțiuni poate fi anulată prin specificarea explicită în linia de comandă a unei valori diferite a opțiunii.

Programele MySQL consultă numeroase fișiere în căutarea opțiunilor; totuși, nu este o eroare dacă un fișier cu opțiuni lipsește. Sub UNIX, sunt explorate în căutarea opțiunilor următoarele fișiere, în ordinea dată:

Numele fișierului	Conținut
`/etc/my.cnf`	Opțiuni globale
`DATADIR/my.cnf`	Opțiuni specifice serverului
`~/.my.cnf`	Opțiuni specifice utilizatorului

Sub Windows, vor fi citite următoarele fișiere:

Numele fișierului	Conținut
`SYSTEMDIR\my.ini`	Opțiuni globale
`C:\my.cnf`	Opțiuni specifice serverului
`DATADIR\my.cnf`	Opțiuni specifice utilizatorului

`DATADIR` reprezintă calea de acces spre catalogul de date din calculatorul dumneavoastră. (Aceasta este calea de acces compilată în server și nu poate fi modificată cu opțiunea `--datadir`.) Sub Windows, `DATADIR` este `C:\mysql\data`. `SYSTEMDIR` reprezintă calea de acces spre catalogul de sistem din Windows (de regulă ceva de genul `C:\Windows` sau `C:\WinNT`). Deși numele fișierelor cu opțiuni din Windows sunt specificate folosind `\` ca separator între componentele căii de acces, căile de acces specificate în cadrul fișierelor trebuie scrise folosind caracterul `/` drept separator. Dacă doriți să folosiți `\` ca separator, folosiți `\\` deoarece `\` este tratat drept caracter escape.

Fișierele cu opțiuni globale sunt folosite de către toate programele care recunosc fișierele cu opțiuni. Un fișier din catalogul de date al unui server este folosit numai de către serverul corespunzător. Fișierele specifice utilizatorilor sunt folosite de programe rulate de utilizatorul respectiv.

Opțiunile sunt specificate în grupuri. Iată un exemplu:

```
[client]
user=paul
password=secret

[mysql]
no-auto-rehash

[mysqlshow]
status
```

Numele grupurilor sunt date între paranteze drepte și corespund, de obicei, unui nume de program. În exemplul anterior, `[mysql]` indică grupul de opțiuni pentru clientul `mysql`, iar `[mysqlshow]` indică grupul de opțiuni pentru `mysqlshow`. Numele special de grup `[client]` vă permite să specificați opțiuni care se aplică tuturor programelor client. Programele client MySQL standard examinează atât grupul `[client]`, cât și grupul cu același nume ca al clientului. De exemplu, `mysql` examinează grupurile `[client]` și `[mysql]`, iar `mysqlshow` examinează grupurile `[client]` și `[mysqlshow]`.

Toate opțiunile care urmează după numele unui grup sunt asociate cu grupul respectiv. Un fișier cu opțiuni poate conține orice număr de grupuri, iar grupurile prezentate mai târziu au prioritate față de grupurile prezentate anterior. Dacă o opțiune dată apare de mai multe ori în grupurile pe care le examinează un program, se va folosi valoarea enumerată ultima.

Fiecare opțiune trebuie scrisă pe o linie separată. Primul cuvânt din linie este numele opțiunii, care trebuie specificat într-un format de tip nume lung, fără cratimele de început. (De exemplu, pentru a preciza opțiunea de comprimare în linia de comandă, puteți folosi `-C` sau `--compress`, dar într-un fișier cu opțiuni puteți folosi numai `compress`.) Orice opțiune în format lung acceptată de un program poate fi enumerată într-un fișier cu opțiuni. Dacă opțiunea necesită o valoare, specificați numele și valoarea separate printr-un caracter =.

Să examinăm următoarea linie de comandă:

```
mysql --compress --user=paul --set-variable max_allowed_packet=24M
```

Pentru a specifica aceleași informații într-un fișier cu opțiuni folosind grupul `[mysql]`, puteți proceda astfel:

```
[mysql]
compress
user=paul
set-variable=max_allowed_packet=24M
```

Observați că, într-un fișier cu opțiuni, `set-variable` este urmată de un caracter `=` în afara caracterului = dintre numele variabilei și valoarea acesteia.

Liniile vide ale fișierelor cu opțiuni sau liniile care încep cu unul din caracterele # sau ; sunt tratate drept comentarii și ignorate.

În valorile opțiunilor se pot folosi anumite secvențe escape pentru a indica anumite caractere speciale:

Secvență	Semnificație
`\b`	Ștergerea caracterului anterior cursorului
`\n`	Linie nouă
`\r`	Retur de car
`\s`	Spațiu
`\t`	Tabulator
`\\`	Backslash

Fiți atent să nu inserați în grupul `[client]` opțiuni care pot fi înțelese numai de către un singur client. De exemplu, dacă inserați în grupul `[client]` opțiunea `skip-line-numbers` specifică clientului `mysql`, veți descoperi subit că alte programe client, precum `mysqlimport`, nu mai funcționează. (Veți primi un mesaj de eroare, urmat de mesajul de asistență.) Mutați opțiunea `skip-line-numbers` în grupul `[mysql]` și totul va fi în regulă.

Opțiunile acceptate de către toate programele MySQL care citesc grupul `[client]` sunt `compress`, `debug`, `help`, `host`, `password`, `pipe`, `port`, `select`, `user` și `version`. (Nu are nici un sens să inserați `help` sau `version` într-un fișier cu opțiuni, desigur.)

Păstrați confidențialitatea fișierelor cu opțiuni specifice utilizatorului

Verificați ca fiecare fișier cu opțiuni specific utilizatorului să se afle în posesia utilizatorului adecvat și ca modul este configurat la 600 sau 400, astfel încât fișierul să nu poată fi citit de alți utilizatori. Este de dorit ca informațiile privind numele de utilizator și parola să nu fie accesibile nici unei alte persoane în afara utilizatorului pentru care este valabil fișierul.

Variabile de mediu

Programele MySQL examinează valorile a numeroase variabile de mediu pentru a obține valorile opțiunilor. Variabilele de mediu au o precedență redusă; opțiunile specificate folosind variabile de mediu pot fi anulate prin opțiuni specificate într-un fișier cu opțiuni sau în linia de comandă.

Programele MySQL verifică următoarele variabile de mediu.

- `MYSQL_DEBUG`

  Opțiunile de utilizat la depanare. Această variabilă nu are nici un efect dacă sistemul MySQL nu a fost construit cu suportul pentru depanare activat. Configurarea variabilei `MYSQL_DEBUG` este similară cu utilizarea opțiunii `--debug`.

- `MYSQL_PWD`

  Parola care se va folosi pentru stabilirea conexiunilor la serverul MySQL. Configurarea variabilei `MYSQL_PWD` este similară cu utilizarea opțiunii `--password`.

- `MYSQL_TCP_PORT`

  Pentru clienți, acesta este numărul de port care se va folosi la stabilirea unei conexiuni TCP/IP cu serverul. Pentru `mysqld`, acesta este portul la care serverul va aștepta conexiunile TCP/IP. Configurarea variabilei `MYSQL_TCP_PORT` este similară cu utilizarea opțiunii `--port`

- `MYSQL_UNIX_PORT`

  Pentru clienți, aceasta este calea de acces a fișierului soclu folosit la stabilirea conexiunilor cu socluri de domeniu UNIX la serverul care rulează pe `localhost`. Pentru `mysqld`, acesta este soclul la care se așteaptă conexiuni locale. Configurarea variabilei `MYSQL_UNIX_PORT` este similară cu folosirea opțiunii `--socket`.
- `TMPDIR`

  Calea de acces la catalogul în care se vor crea fișierele temporare. Configurarea acestei variabile este similară cu utilizarea opțiunii `--tmpdir`.
- `USER`

  Acesta este numele de utilizator MySQL care se va folosi pentru conectarea la server. Această variabilă este folosită numai de către programele client care rulează sub Windows; configurarea acesteia este similară cu utilizarea opțiunii `--user`.

  Clientul `mysql` verifică valoarea următoarelor variabile de mediu:
- `MYSQL_HISTFILE`

  Numele fișierului care va fi folosit pentru stocarea istoricului utilizării liniei de comandă. Valoarea prestabilită a acestei variabile, dacă nu este configurată, este `$HOME/.mysql_history`, unde `$HOME` este amplasamentul catalogului dumneavoastră de bază.
- `MYSQL_HOST`

  Gazda la care se va conecta clientul atunci când stabilește o conexiune cu serverul. Configurarea acestei variabile este similară cu utilizarea opțiunii `--host`.

`isamchk` și `myisamchk`

Aceste utilitare vă permit să verificați și să reparați tabelele deteriorate, să afișați informații despre tabele, să executați analiza distribuției valorilor cheie din index și să dezactivați, respectiv să activați indexurile. Capitolul 4, „Optimizarea interogărilor", conține mai multe informații cu privire la analiza cheilor și la dezactivarea indexurilor. Capitolul 13, „Întreținerea și repararea bazelor de date", furnizează mai multe informații despre verificarea și repararea tabelelor.

`myisamchk` este utilizat pentru tabele care folosesc formatul de stocare mai recent MyISAM. Aceste tabele prezintă nume de fișiere de date, respectiv index, cu sufixele `.MYD` și `.MYI`. Utilizați `isamchk` pentru tabele care folosesc mai vechiul format de stocare ISAM. Acestea folosesc nume de fișiere de date, respectiv index, cu sufixele `.ISD` și `.ISM`. Dacă indicați oricărui utilitar să opereze pe un tabel de un alt tip decât cel căruia îi este destinat, programul va afișa un mesaj de avertizare și va ignora tabelul.

Nu executați verificări sau reparări ale tabelelor în timp ce serverul MySQL are acces la tabel! Capitolul 13 discută despre metodele de a împiedica serverul să folosească un tabel în timp ce `isamchk` sau `myisamchk` lucrează cu tabelul respectiv.

Utilizare

```
isamchk [optiuni] nume_tabel[.ISM] ...
myisamchk [optiuni] nume_tabel[.MYI] ...
```

Dacă nu sunt specificate opțiuni, aceste utilitare verifică tabelele denumite, în căutarea erorilor. În caz contrar, tabelele sunt prelucrate în conformitate cu semnificația opțiunilor specificate. Un argument `nume_tabel` poate fi numele unui tabel sau numele fișierului index aferent tabelului (`.ISM` sau `.MYI`). Este convenabilă utilizarea numelor de fișiere index, deoarece puteți folosi caractere de înlocuire pentru nume de fișiere pentru ca programul să acționeze dintr-o singură comandă asupra tuturor tabelelor de un anumit tip de stocare. De exemplu, puteți verifica toate tabelele dintr-un catalog în acest mod:

```
% isamchk *.ISD
% myisamchk *.MYI
```

Aceste utilitare nu fac nici o presupunere cu privire la amplasamentul fișierelor tabel. Trebuie să specificați calea de acces spre fișierele pe care doriți să le folosiți, dacă acestea nu se află în catalogul curent. Deoarece se presupune că fișierele tabel nu sunt amplasate în catalogul de date al serverului, puteți copia fișierele tabel într-un alt catalog și puteți lucra folosind copiile în locul originalelor.

Unele dintre opțiuni se referă la numere de index. Indexurile sunt numerotate începând de la 1. Puteți emite o interogare `SHOW INDEX` sau puteți folosi comanda `mysqlshow --keys` pentru a descoperi numerotarea de index pentru un anumit tabel. Coloana `Key_name` afișează indexurile în aceeași ordine în care cele două programe „văd" indexurile respective.

> **MYSQL_PWD este un pericol la adresa securității**
>
> Utilizarea variabilei `MYSQL_PWD` pentru stocarea unei parole se constituie într-un risc de securitate, deoarece alți utilizatori ai sistemului dumneavoastră pot descoperi cu ușurință valoarea sa. De exemplu, utilitarul ps va afișa configurațiile variabilelor de mediu pentru alți utilizatori.

Opțiuni standard acceptate de `isamchk` și `myisamchk`

`--debug`	`--set-variable`	`--verbose`
`--help`	`--silent`	`--version`

Opțiunea `--silent` arată că vor fi afișate numai mesajele de eroare. Opțiunea `--verbose` afișează mai multe informații, atunci când este furnizată cu opțiunile `--description` sau `--extend-check` pentru oricare din cele două programe, respectiv când este dată cu opțiunea `--check` pentru `myisamchk`. Opțiunile `--silent` și `--verbose` pot fi specificate de mai multe ori, pentru a le spori efectul.

Opțiuni comune programelor `isamchk` și `myisamchk`

Cele două utilitare au numeroase opțiuni în comun:

- `-a, --analyze`

  Execută analiza distribuției cheilor. Această operație poate ajuta serverul să execute mai rapid căutări și uniri bazate pe index. Puteți obține informații despre distribuția

cheilor în urma analizei prin rularea din nou a utilitarelor `isamchk` sau `myisamchk`, cu opțiunile `--description` și `--verbose`.

- `-b n, --block-search=n`

  Afișează începutul rândului de tabel care conține un bloc, începând de la blocul *n*. Această opțiune se folosește numai la depanare.

- `-d, --description`

  Afișează informații semnificative referitoare la tabel.

- `-e, --extend-check`

  Execută o verificare amănunțită a tabelului. Utilizarea acestei opțiuni este rareori necesară, deoarece `myisamchk` și `isamchk` pot găsi toate erorile folosind unul din modurile de verificare mai puțin aprofundată.

- `-f, --force`

  Forțează verificarea sau repararea unui tabel chiar și atunci când există deja un fișier temporar pentru tabel. În mod normal, `myisamchk` și `isamchk` pur și simplu își încheie execuția după afișarea unui mesaj de eroare dacă găsesc un fișier *nume_tabel*`.TMD`, deoarece acest lucru poate însemna că o altă instanță a programului este deja în funcțiune. Totuși, fișierul mai poate exista și dacă ați suprimat o invocare anterioară a programului în timp ce acesta funcționa, caz în care fișierul poate fi eliminat fără riscuri. Dacă știți că aceasta este situația, folosiți opțiunea `--force` pentru indica oricăruia dintre utilitare să ruleze chiar dacă există un fișier temporar. (Alternativ, puteți elimina manual fișierul temporar.)

  Dacă folosiți opțiunea `--force` la verificarea tabelelor, programul va reporni automat cu opțiunea `--recover` pentru fiecare tabel unde au fost depistate probleme.

- `-i, --information`

  Afișează informații cu caracter statistic relativ la conținutul tabelului.

- `-k n, --keys-used=n`

  Se folosește cu opțiunea `--recover`. Indică programului MySQL să actualizeze numai primele *n* indexuri. Cu alte cuvinte, sunt dezactivate indexurile cu numere mai mari decât *n*. Această opțiune se poate folosi pentru îmbunătățirea performanțelor operațiilor `INSERT`, `DELETE` și `UPDATE`. Prin specificarea unui număr de index egal cu 0 se dezactivează toate indexurile. Prin specificarea unui număr de indexuri egal cu numărul de index cel mai mare se reactivează toate indexurile.

- `-l, --no-symlinks`

  Dacă un argument tabel este o legătură simbolică, în mod normal va fi reparat tabelul spre care indică legătura simbolică. Când este specificată această opțiune, legăturile simbolice sunt ignorate. În schimb, legătura simbolică este *înlocuită* cu versiunea nouă (reparată) a fișierului.

- `-q, --quick`

  Această opțiune este folosită în conjuncție cu `--recover` pentru o reparare mai rapidă decât atunci când se utilizează numai `--recover`. Fișierul de date este neatins atunci când sunt specificate ambele opțiuni. Pentru a forța programul să modifice fișierul de date chiar și atunci când sunt găsite valori duplicate ale cheilor, specificați opțiunea `--quick` de două ori. Dacă folosiți această opțiune de două ori, nu uitați să executați în prealabil o copie de siguranță a fișierului de date.

- `-r, --recover`

  Execută o operație normală de refacere. Aceasta poate remedia majoritatea problemelor, cu excepția valorilor duplicate aflate într-un index care ar trebui să fie unic.

- `-o, --safe-recover`

  Folosește o metodă de refacere mai lentă decât metoda `--recover`, dar care poate rezolva câteva probleme imposibil de remediat de către aceasta din urmă.

- `-S, --sort-index`

  Sortează blocurile de index, pentru accelerarea operațiilor de citire secvențială a blocurilor.

- `-R n, --sort-records=n`

  Sortează înregistrările de date conform ordinii în care sunt afișate înregistrările în indexul *n*. Operațiile ulterioare de regăsire bazate pe indexul dat vor avea o viteză mai mare. Prima dată când executați această operație asupra unui tabel, viteza de execuție poate fi foarte redusă, deoarece înregistrările dumneavoastră nu vor fi așezate în ordine.

- `-u, --unpack`

  Despachetează un fișier împachetat. `myisamchk` poate despacheta fișiere MyISAM împachetate cu `myisampack`; `isamchk` poate despacheta fișiere ISAM împachetate cu `pack_isam`. Această opțiune se poate folosi la conversia într-o formă modificabilă a unui tabel comprimat numai pentru citire.

- `-w, --wait`

  Dacă un tabel este blocat, așteaptă până când devine disponibil. Fără opțiunea `--wait`, programul va aștepta o blocare timp de 10 secunde și apoi va afișa un mesaj de eroare dacă nu poate obține nici o blocare.

Opțiuni specifice utilitarului `myisamchk`

Deși nu există opțiuni specifice programului `isamchk`, există numeroase opțiuni caracteristice utilitarului `myisamchk`.

- `-c, --check`

  Verifică tabelele în căutarea erorilor. Aceasta este acțiunea prestabilită dacă nu sunt specificate opțiuni.

- `-D n, --data-file-length=n`

  Specifică lungimea maximă permisă a fișierului de date atunci când se reconstruiește un fișier de date care s-a umplut. (Acest lucru se produce când un fișier atinge maxima dimensională impusă de MySQL sau de limitările impuse de sistemul dumneavoastră de operare asupra dimensiunilor fișierelor. De asemenea, se produce atunci când numărul de rânduri atinge limita impusă de structurile de date interne ale tabelelor.)

Valoarea este specificată în octeți. Această opțiune are efect numai atunci când este folosită cu --recover sau --safe-recover.

- -F, --fast

  Nu verifică tabelele care nu au suferit modificări de la ultima verificare.

- -m, --medium-check

  Verifică un tabel folosind o metodă mai rapidă decât metoda de verificare aprofundată, dar ceva mai puțin „pătrunzătoare" decât aceasta. (Mesajul de asistență al utilitarului myisamchk arată că această metodă găsește „numai" 99,99% din erori.) Acest mod de verificare este suficient în majoritatea situațiilor. Modul de verificare mediu lucrează prin calculul valorilor CRC ale cheilor din index și compararea acestora cu valorile CRC calculate din coloanele indexate din fișierul de date.

- -T, --read-only

  Nu marchează tabelul ca fiind verificat.

- -A [*n*], --set-auto-increment[=*n*]

  Forțează valorile AUTO_INCREMENT să înceapă de la *n* (respectiv de la o valoare mai mare, dacă tabelul conține deja înregistrări cu valorile AUTO_INCREMENT egale cu *n*). Dacă nu este specificată nici o valoare, această opțiune configurează următoarea valoare AUTO_INCREMENT astfel încât să fie cu o unitate mai mare decât valoarea curentă maximă.

- -t *nume_cale*, --tmpdir=*nume_cale*

  Precizează calea de acces a catalogului care se va utiliza pentru fișierele temporare. Valoarea prestabilită este valoarea variabilei de mediu sau /tmp dacă variabila respectivă nu este configurată.

Variabile pentru isamchk și myisamchk

Următoarele variabile folosite de utilitarele isamchk și myisamchk pot fi configurate cu opțiunea --set-variable:

- key_buffer_size

  Dimensiunea bufferului folosit pentru blocurile de index.

- read_buffer_size

  Dimensiunea bufferului de citire.

- write_buffer_size

  Dimensiunea bufferului de scriere.

- sort_buffer_size

  Dimensiunea bufferului folosit pentru operațiile de sortare după valorile cheilor. (Variabila este folosită pentru opțiunea --recover, nu pentru --safe-recover.)

- sort_key_blocks

  Această variabilă este corelată cu adâncimea structurii arborelui B folosit pentru index. Nu este necesar să o modificați.

- decode_bits

  Numărul de biți care se va folosi la decodarea tabelelor comprimate. Valori mai mari ale acestui parametru pot avea ca rezultat o operare mai rapidă, dar vor necesita o cantitate suplimentară de memorie. În general, valoarea prestabilită este suficientă.

myisampack și pack_isam

Utilitarele myisampack și pack_isam produc tabele comprimate, numai pentru citire. Acestea realizează reduceri caracteristice ale spațiului de stocare necesar cuprinse între 40 și 70 de procente, păstrând însă o viteză mare de acces la înregistrări. Utilitarul myisampack împachetează tabelele MyISAM și funcționează cu toate tipurile de coloane. Programul pack_isam împachetează tabelele ISAM și funcționează numai cu tabele care nu conțin coloane BLOB sau TEXT.

myisampack și pack_isam nu sunt incluse în MySQL în mod prestabilit, dar sunt disponibile pentru clienții care comandă minimum 10 licențe sau care beneficiază de un contract de asistență la nivelul de asistență extins sau la un nivel superior. Aceste prevederi se pot modifica astfel încât programele să fie disponibile în aceleași condiții ca și celelalte componente ale distribuției MySQL. (Vezi manualul de referință MySQL pentru informații curente privind asistența și licențele.)

Pentru citirea tabelelor care au fost împachetate cu aceste utilitare nu este necesară nici o versiune specială a sistemului MySQL, ceea ce face aceste programe utilizabile mai ales în aplicații pentru care dumneavoastră doriți să distribuiți un tabel conținând informații de arhivă sau de natură enciclopedică numai pentru citire și care nu trebuie actualizate. Dacă doriți să convertiți un fișier împachetat în forma despachetată și implicit modificabilă, puteți proceda astfel folosind myisamchk --unpack (pentru tabelele MyISAM), respectiv isamchk --unpack (pentru tabelele ISAM).

myisampack și pack_isam împachetează fișierele de date, dar nu se ating de fișierele index. Pentru a actualiza indexurile, rulați myisamchk -rq după ce ați rulat myisampack sau isamchk -rq, după ce ați rulat pack_isam.

Utilizare

```
myisampack [optiuni] nume_tabel ...
pack_isam [optiuni] nume_tabel ...
```

Un argument *nume_tabel* poate fi numele unui tabel sau numele fișierului index aferent tabelului (un fișier .MYI pentru tabelele MyISAM, respectiv un fișier .ISM pentru tabelele ISAM). Specificați calea de acces la catalogul în care este localizat tabelul, dacă nu vă aflați în catalogul respectiv.

Opțiuni standard acceptate de myisampack și pack_isam

--debug	--silent	--version
--help	--verbose	

Opțiuni comune programelor myisampack și pack_isam

- `-b, --backup`

  Indică utilitarelor `myisampack` sau `pack_isam` să execute o copie de siguranță pentru fiecare argument *nume_tabel* sub numele *nume_tabel*`.OLD` înainte de a-l împacheta.

- `-f, --force`

  Forțează împachetarea unui tabel chiar dacă fișierul împachetat rezultant este mai mare decât originalul sau dacă există deja un fișier temporar aferent tabelului. În mod normal, `myisampack` și `pack_isam` își încheie pur și simplu execuția după afișarea unui mesaj de eroare, dacă găsesc un fișier cu numele *nume_tabel*`.TMD`, deoarece acesta poate fi un indiciu că o altă instanță a programului este deja în funcțiune. Cu toate acestea, fișierul poate exista și dacă ați suprimat o invocare anterioară a programului în timpul rulării acestuia, caz în care fișierul poate fi eliminat fără probleme. Dacă știți că aceasta este situația, folosiți `--force` pentru a indica oricărui utilitar să împacheteze tabelul, chiar dacă fișierul temporar există.

- `-j` *unire_tbl*, `--join=`*unire_tbl*

  Unește (fuzionează) toate tabelele denumite în linia de comandă într-un singur tabel împachetat, denumit *unire_tbl*. Toate tabelele care vor fuziona trebuie să aibă aceeași structură. (Numele coloanelor, tipurile și indexurile trebuie să fie identice.)

- `-t, --test`

  Rulează în modul test. Este rulat un test de împachetare și se afișează informații despre rezultatele pe care le-ați obține dacă ați împacheta efectiv tabelul.

- `-T` *nume_cale*, `--tmpdir=`*nume_cale*

  Precizează numele căii de acces la catalogul care se va utiliza pentru fișierele temporare.

- `-W, --wait`

  Așteaptă și reia încercarea dacă există vreun tabel în uz. Nu doriți să împachetați un tabel dacă acesta poate fi actualizat în timp ce îl împachetați.

Opțiuni specifice utilitarului pack_isam

Deși nu există opțiuni specifice pentru `myisampack`, există una specifică utilitarului `pack_isam`, și anume `-p` *n*, `--packlength=`*n*. Această opțiune indică programului `pack_isam` să folosească un spațiu de stocare a lungimii înregistrării egal cu *n* octeți, unde *n* este un întreg cuprins între 1 și 3. Utilitarul `pack_isam` încearcă în mod automat să determine numărul octeților necesari pentru înregistrarea lungimii fiecărei înregistrări din tabelul împachetat. În unele situații, poate ajunge la concluzia că este posibilă utilizarea unei lungimi mai mici, caz în care `pack_isam` emite un mesaj în acest sens. Puteți rula `pack_isam` din nou și specificați în mod explicit lungimea mai mică folosind o opțiune `--packlength` pentru a obține unele economii suplimentare de spațiu.

mysql

Clientul `mysql` este un program interactiv care vă permite să vă conectați la server, s emiteți interogări și să vizualizați rezultatele. De asemenea, `mysql` mai poate fi folosit î modul batch, pentru a executa interogări stocate într-un fișier, dacă redirecționaț intrarea comenzii astfel încât să citească din fișierul respectiv. De exemplu:

```
% mysql -u paul -p -h pit-viper.snake.net samp_db < my_query_file
```

În modul interactiv, când `mysql` pornește, afișează un prompt `mysql` pentru a arăta c așteaptă datele de intrare. Pentru a emite o interogare, tastați-o (folosind mai multe linii dacă este necesar) și apoi indicați sfârșitul interogării tastând ; (caracterul punct și vir gulă) sau `\g`. `mysql` trimite interogarea la server, afișează rezultatele și apoi afișează u alt prompt, indicând că este pregătit pentru o altă interogare.

În `mysql`, promptul poate lua diferite forme, pentru a indica datele de intrare pe care l așteaptă:

`mysql`	Așteaptă prima linie a unei noi interogări
`->`	Așteaptă linia următoare a interogării curente
`'`	Așteaptă finalizarea unui șir cu ghilimele simple din interogarea curentă
`"`	Așteaptă finalizarea unui șir cu ghilimele duble din interogarea curent

Prompturile `'` și `"` arată că ați început să scrieți un șir cu ghilimele simple, respectiv dubl într-o linie anterioară și nu ați introdus încă ghilimelele de închidere. De regulă, aceast se întâmplă când ați uitat să încheiați un șir. Dacă aceasta este situația, pentru a părăs modul de colectare a șirurilor, introduceți ghilimelele de închidere adecvate indicate d prompt, urmate de expresia `\c` pentru a anula interogarea curentă.

Când programul `mysql` este folosit în modul interactiv, salvează interogările într-u fișier istoric. Acest fișier este `$HOME/.mysql_history` în mod prestabilit sau poate fi spe cificat în mod explicit prin configurarea variabilei de mediu `MYSQL_HISTORY`. Interogăril pot fi reluate din istoricul comenzilor și emise din nou, cu sau fără alte modificări. List următoare prezintă unele dintre aceste comenzi de editare (este posibil ca tastele c săgeți să nu funcționeze sub Windows):

Secvență de taste	Semnificație
Săgeată sus, Ctrl-P	Reia linia anterioară
Săgeată jos, Ctrl-N	Reia linia următoare
Săgeată stânga, Ctrl-B	Mută cursorul la stânga (înapoi)
Săgeată dreapta, Ctrl-F	Mută cursorul la dreapta (înainte)
Escape Ctrl-B	Mută cursorul înapoi cu un cuvânt
Escape Ctrl-F	Mută cursorul înainte cu un cuvânt
Ctrl-A	Mută cursorul la începutul rândului
Ctrl-E	Mută cursorul la sfârșitul rândului
Ctrl-D	Șterge caracterul plasat sub cursor

Secvență de taste	Semnificație
Delete	Șterge caracterul plasat la stânga cursorului
Escape D	Șterge cuvântul
Escape Backspace	Șterge cuvântul plasat la stânga cursorului
Ctrl-K	Șterge toate caracterele de la cursor la sfârșitul liniei
Ctrl-_	Anulează ultima modificare; poate fi repetat

Unele opțiuni nu permit utilizarea fișierului istoric. În general, acestea sunt opțiuni care indică utilizarea non-interactivă a programului `mysql`, precum `--batch`, `--html` și `--quick`.

Utilizare

`mysql [opțiuni] [nume_db]`

Dacă specificați un argument *nume_db*, baza de date respectivă devine baza de date curentă (prestabilită) pentru sesiunea dumneavoastră. Dacă nu specificați un asemenea argument, `mysql` începe fără nici o bază de date curentă și va trebui fie să completați toate referințele la tabele cu numele unei baze de date, fie să emiteți o instrucțiune `USE` *nume_db* pentru a specifica o bază de date prestabilită.

Opțiuni standard acceptate de mysql

`--compress`	`--password`	`--silent`	`--version`
`--debug`	`--pipe`	`--socket`	
`--help`	`--port`	`--user`	
`--host`	`--set-variable`	`--verbose`	

Opțiunile `--silent` și `--verbose` pot fi specificate de mai multe ori, pentru a se obține un efect sporit al acestora.

Opțiuni specifice programului mysql

- `-B, --batch`

  Precizează că `mysql` trebuie să ruleze în modul batch. Rezultatele interogărilor sunt afișate într-un format delimitat prin tabulatori (fiecare rând pe o linie separată, cu tabulatori între valorile din coloane). Acest lucru este convenabil în special pentru generarea datelor de ieșire pe care doriți să le importați într-un alt program, precum o foaie de calcul tabelar. Rezultatele interogărilor includ, în mod prestabilit, un rând inițial de antete de coloană. Pentru a elimina aceste antete, folosiți opțiunea `skip-column-names`.

- `-T, --debug-info`

  Afișează informații de depanare atunci când programul își încheie execuția.

- `-e interogare, --execute=interogare`

  Execută interogarea și își încheie execuția. Trebuie să includeți interogarea între ghilimele, pentru a preveni interpretarea acesteia de către shell sub formă de multiple argumente în linia de comandă. Se pot specifica mai multe interogări, separându-le prin caractere punct și virgulă în șirul `interogare`.

- `-f, --force`

  În mod normal, când `mysql` citește interogări dintr-un fișier, își încheie execuția dacă se produce vreo eroare. Această opțiune determină programul `mysql` să continue prelucrarea interogărilor, indiferent de erori.

- `-H, --html`

  Generează date de ieșire în format HTML. Opțiunea a devenit funcțională în MySQL 3.22.26.

- `-i, --ignore-space`

  Indică serverului să ignore spațiile dintre numele funcțiilor și caracterul paranteză rotundă de deschidere care inițiază lista cu argumente. În mod normal, numele funcțiilor trebuie să fie urmate imediat de către paranteze, fără spații intermediare.

- `-A --no-auto-rehash`

  Când `mysql` pornește, calculează valorile hash ale numelor bazelor de date, tabelelor și coloanelor pentru a construi o structură de date care permite completarea rapidă a numelor. (Puteți tasta partea inițială a unui nume atunci când introduceți o interogare și apoi apăsați pe tasta Tab; `mysql` va completa numele, dacă nu există ambiguități.) Această opțiune elimină calculul valorilor hash, ceea ce permite programului `mysql` să pornească mai rapid, în special dacă dispuneți de multe tabele. Dacă doriți să folosiți caracteristica de completare a numelor după pornirea programului `mysql`, puteți folosi comanda `rehash` la promptul `mysql`.

- `-o --one-database`

  Această opțiune este folosită când actualizați baze de date pornind de la conținutul unui fișier jurnal de actualizare. Opțiunea indică programului MySQL să actualizeze numai baza de date prestabilită (baza de date denumită în linia de comandă). Actualizările aduse altor baze de date sunt ignorate. Dacă nici o bază de date nu este denumită în linia de comandă, nu sunt executate actualizări.

- `-q, --quick`

  În mod normal, `mysql` regăsește integral rezultatul unei interogări de la server înainte de a-l afișa. Această opțiune determină afișarea fiecărui rând pe măsură ce rândul respectiv este regăsit, utilizându-se astfel o cantitate mult mai redusă de memorie și permițând executarea cu succes a unor interogări de mari dimensiuni, care în alte condiții ar fi eșuat. Totuși, această opțiune nu trebuie specificată pentru uz interactiv; dacă utilizatorul oprește temporar datele de ieșire sau suspendă programul `mysql`, serverul continuă să aștepte, ceea ce poate afecta funcționarea altor clienți.

- `-r, --raw`

  Scrie valorile coloanelor fără a modifica semnificația eventualelor caractere speciale. Această opțiune este folosită în conjuncție cu opțiunea `--batch`.

- `-N, --skip-column-names`

  Inhibă afișarea numelor de coloane ca antete de coloană în rezultatele interogărilor. Puteți obține acest efect și prin specificarea de două ori a opțiunii `--silent`.

- `-L, --skip-line-numbers`

Nu scrie numerele liniilor atunci când se produc erori.

- `-t, --table`

  Produce datele de ieşire în format tabular, cu valorile din fiecare rând delimitate de bare şi aliniate pe verticală.

- `-n, --unbuffered`

  Goleşte după fiecare interogare bufferul folosit pentru comunicaţii cu serverul.

- `-E, --vertical`

  Afişează rezultatele interogării pe verticală, adică fiecare rând al unui set de rezultate este afişat sub formă de set de linii, câte o linie pentru fiecare coloană. Fiecare linie constă din numele şi valoarea unei coloane. Afişarea fiecărui rând este precedată de o linie care indică numărul rândului din cadrul setului de rezultate. Formatul de afişare pe verticală poate fi util atunci când o interogare generează linii foarte lungi.

  Dacă această opţiune nu este specificată, puteţi activa formatul de afişare pe verticală pentru interogările individuale prin utilizarea caracterului de terminare `\G`, în loc de `;` sau `\g`.

  Această opţiune a fost introdusă în versiunea MySQL 3.22.5.

- `-w, --wait`

  Dacă o conexiune cu serverul nu poate fi stabilită, aşteaptă şi reia încercarea.

Variabile pentru mysql

Următoarele variabile pentru `mysql` pot fi configurate cu opţiunea `--set-variable`:

- `max_allowed_packet`

  Dimensiunea maximă a bufferului folosit pentru comunicaţia între server şi client.

- `net_buffer_length`

  Dimensiunea iniţială a bufferului folosit pentru comunicaţia între server şi client. Acest buffer poate fi extins până la maximum `max_allowed_packet` octeţi.

Comenzi folosite în mysql

Programul `mysql` înţelege numeroase comenzi şi vă permite să expediaţi instrucţiuni SQL serverului MySQL. Fiecare comandă trebuie emisă pe o singură linie; la sfârşitul liniei nu este necesar caracterul punct şi virgulă. Majoritatea comenzilor au o formă lungă, alcătuită dintr-un cuvânt, respectiv o formă scurtă, care constă dintr-un caracter backslash urmat de o singură literă. Comenzile în forma lungă nu sunt sensibile la diferenţa între majuscule şi minuscule. Comenzile în formă scurtă trebuie specificate folosind dimensiunile literelor prezentate în lista următoare:

- `clear, \c`

  Şterge (anulează) interogarea curentă. Interogarea curentă este acea interogare pe care o tastaţi la un moment dat; această comandă nu anulează o interogare care a fost deja trimisă serverului şi pentru care `mysql` afişează date de ieşire.

- `connect [nume_db [nume_gazda]], \r [nume_db [nume_gazda]]`

  Se conectează la baza de date specificată sau la gazda menţionată. Dacă numele bazei de date sau numele gazdei lipsesc, se vor utiliza valorile cel mai recent folosite din sesiunea `mysql` curentă.

- `edit, \e`

  Editează interogarea curentă. Editorul care se va utiliza este determinat de valoarea variabilelor de mediu `EDITOR` şi `VISUAL`, respectiv este `vi` dacă nici una dintre acestea nu este configurată.

- `ego, \G`

  Trimite interogarea curentă la server şi afişează rezultatul pe verticală. Această comandă a fost introdusă în versiunea MySQL 3.22.11.

- `exit`

  Similară cu `quit`.

- `go, \g, ;`

  Trimite interogarea curentă la server şi afişează rezultatul.

- `help, \h, ?`

  Afişează un mesaj de asistenţă, care descrie comenzile `mysql` disponibile.

- `print, \p`

  Afişează interogarea curentă (textul interogării în sine, nu rezultatele obţinute prin executarea interogării).

- `quit, \q`

  Încheie execuţia programului `mysql`.

- `rehash, \#`

  Recalculează informaţiile necesare pentru completarea numelor bazelor de date, ale tabelelor şi ale coloanelor. Vezi descrierea opţiunii `--no-auto-rehash`.

- `status, \s`

  Regăseşte şi afişează informaţiile de stare de la server. Este o opţiune utilă dacă doriţi să verificaţi versiunea serverului şi baza de date curentă.

- `use nume_db, \u nume_db`

  Selectează baza de date specificată pentru a o transforma în bază de date curentă.

mysql.server

mysql.server este un script care pornește și oprește serverul `mysqld` invocând scriptul `safe_mysqld`.

Utilizare

```
mysql.server start
mysql.server stop
```

În mod normal, `mysql.server` este folosit pe un sistem stil System V și este instalat într-unul din cataloagele de rulare din catalogul `/etc`. Sistemul pornește serverul invocând scriptul cu argumentul `start` în momentul încărcării sistemului de operare. Sistemul închide serverul invocând scriptul cu argumentul `stop` în momentul opririi sistemului. De asemenea, scriptul poate fi invocat manual, cu argumentul adecvat, pentru a porni, respectiv opri serverul.

mysqlaccess

Acest script vă permite să vă conectați la un server, să regăsiți informații despre privilegiile de acces și să testați rezultatele specificării privilegiilor de utilizator. Scriptul execută aceste operații folosind copii ale tabelelor `user`, `db` și `host` din baza de date `mysql`. (`mysqlaccess` nu se poate folosi pentru exersarea utilizării privilegiilor la nivel sau de coloană.) De asemenea, puteți înregistra schimbările din tabelele temporare în tabelele efective din baza de date `mysql`.

Pentru a folosi `mysqlaccess`, trebuie să aveți suficiente privilegii pentru a obține personal accesul la tabelele de acordare.

Utilizare

`mysqlaccess [`*`nume_gazda`*` [`*`nume_utilizator`*` [`*`nume_db`*`]]] `*`optiuni`*

Opțiuni standard acceptate de mysqlaccess

`--host` `--password` `--user` `--version`

Opțiuni specifice programului mysqlaccess

- `-b, --brief`

  Afișează rezultatele în format cu o singură linie.
- `--commit`

  Copiază tabelele de acordare temporare înapoi în baza de date `mysql`. Nu uitați să executați ulterior o comandă `mysqladmin flush-privileges`, astfel încât serverul să remarce schimbările.
- `--copy`

  Încarcă tabelele de acordare în tabelele temporare.

- `-d` *`nume_db`*`, --db=`*`nume_db`*

  Specifică numele bazei de date.
- `--debug=`*`n`*

  Specifică nivelul de depanare. *n* trebuie să fie un întreg cuprins între 0 și 3.
- `--howto`

  Afișează unele exemple privind modul de utilizare a programului `mysqlaccess`.
- `--old-server`

  Arată că serverul este mai vechi decât versiunea MySQL 3.21. Aceasta determină programul `mysqlaccess` să aducă anumite modificări interogărilor pe care le trimite la server.
- `--plan`

  Afișează o listă cu îmbunătățirile planificate pentru versiunile viitoare ale programului `mysqlaccess`.
- `--preview`

  Afișează diferențele în materie de privilegii între tabelele de acordare temporare și efective.
- `--relnotes`

  Prezintă informațiile de lansare ale programului `mysqlaccess`.
- `-H` *`nume_gazda`*`, --rhost=`*`nume_gazda`*

  Specifică gazda situată la distanță a serverului la care se va stabili conexiunea.
- `--rollback`

  Anulează modificările aduse tabelelor de acordare temporare.
- `-P` *`parola`*`, --spassword=`*`parola`*

  Specifică parola pentru super-utilizatorul MySQL (un utilizator cu privilegii suficiente pentru modificarea tabelelor de acordare).
- `-U` *`nume_utilizator`*`, --superuser=`*`nume_utilizator`*

  Specifică numele de utilizator al super-utilizatorului MySQL.
- `-t, --table`

  Afișează rezultatele în format cu tabulatori.

mysqladmin

Utilitarul `mysqladmin` comunică cu serverul MySQL pentru a executa o varietate de operații administrative. Puteți folosi `mysqladmin` pentru a obține informații de la server sau pentru a controla operarea serverului, pentru a configura parole și pentru a crea sau șterge bazele de date.

Utilizare

```
mysqladmin [optiuni] comanda ...
```

Opțiuni standard acceptate de mysqladmin

--compress	--host	--port	--user
--debug	--password	--silent	--version
--help	--pipe	--socket	

Opțiunea --silent determină programul mysqladmin să-și încheie execuția în liniște, dacă nu se poate conecta la server.

Opțiuni specifice programului mysqladmin

- `-f, --force`

  Această opțiune are două efecte. Mai întâi, determină programul mysqladmin să nu ceară confirmare pentru comanda `drop nume_db`. În al doilea rând, când sunt specificate mai multe comenzi în linia de comandă, mysqladmin încearcă să execute fiecare comandă, chiar dacă se produc erori. În mod normal, programul mysqladmin își va încheia execuția după prima eroare.

- `--relative`

  Afișează diferența dintre valoarea curentă și valoarea anterioară atunci când se folosește cu --sleep. În prezent, această opțiune funcționează numai alături de comanda extended-status.

- `-i n, --sleep=n`

  Execută în mod repetat comenzile indicate în linia de comandă, cu un interval de *n* secunde între fiecare repetiție.

- `-t n, --timeout=n`

  Așteaptă *n* secunde înainte de a expira atunci când încearcă să se conecteze la server. Această opțiune a fost introdusă în MySQL versiunea 3.22.1.

- `-w [n], --wait[=n]`

  Specifică de câte ori programul trebuie să aștepte și să încerce din nou dacă nu a reușit să stabilească o conexiune cu serverul. Valoarea prestabilită a lui *n* este 1, dacă nu este specificată nici o valoare.

Comenzile din programul mysqladmin

După opțiunile din linia de comandă, puteți specifica una sau mai multe din următoarele comenzi. Numele comenzii poate fi redus la un prefix, cu condiția ca prefixul să fie unic. De exemplu, comanda processlist poate fi redusă la process sau proc, dar nu la forma p.

Multe din aceste comenzi au un echivalent SQL. Acestea sunt notate în cadrul descrierilor. Pentru mai multe informații cu privire la semnificația instrucțiunilor SQL, vezi Anexa D, „Referință de sintaxă SQL“.

- `create nume_db`

  Creează o bază de date nouă, cu numele dat.

  Această comandă este similară comenzii SQL CREATE DATABASE *nume_db*.

- `drop nume_db`

  Șterge baza de date cu numele dat, precum și toate tabelele care se găsesc în baza de date. Fiți atent cu această comandă; nu vă puteți recupera baza de date după ce ați șters-o. Dacă nu a fost menționată opțiunea --force, programul mysqladmin cere confirmarea pentru execuția comenzii.

  Această comandă este similară comenzii SQL DROP DATABASE *nume_db*.

- `extended-status`

  Afișează numele și valorile variabilelor de stare ale serverului.

  Această comandă este similară cu instrucțiunea SQL SHOW STATUS.

  Comanda extended-status a fost introdusă în MySQL 3.22.10.

- `flush-hosts`

  Golește zona cache aferentă gazdelor.

  Această comandă este similară instrucțiunii SQL FLUSH HOSTS.

- `flush-logs`

  Golește (închide și redeschide) fișierele jurnal.

  Această comandă este similară instrucțiunii SQL FLUSH LOGS.

- `flush-privileges`

  Reîncarcă tabelele de acordare.

  Această comandă este similară instrucțiunii SQL FLUSH PRIVILEGES.

- `flush-status`

  Șterge variabilele de stare. (Astfel, numeroase contoare sunt reinițializate la valoarea zero.)

  Această comandă este similară instrucțiunii SQL FLUSH STATUS.

- `flush-tables`

  Golește zona cache aferentă tabelelor.

  Această comandă este similară instrucțiunii SQL FLUSH TABLES.

- `kill id,id ...`

  Suprimă firele de server specificate de numerele de identificator date. Dacă specificați mai multe numere, lista de identificatori nu trebuie să conțină spații, astfel încât să nu fie confundată cu alte comenzi după comanda kill. Pentru a identifica firele ce rulează la un moment dat, folosiți mysqladmin processlist.

  Această comandă este similară unei instrucțiuni SQL KILL pentru fiecare identificator de fir.

- `password parola_noua`

  Modifică parola aferentă utilizatorului MySQL sub al cărui cont v-ați conectat atunci când rulați comanda `mysqladmin`. Noua parolă este *parola_noua*. Trebuie să vă puteți conecta la server sub contul utilizatorului MySQL a cărui parolă este modificată, ceea ce servește ca o confirmare a faptului că știți vechea parolă a utilizatorului.

  Această comandă este similară instrucțiunii SQL `SET PASSWORD`.

  Comanda `PASSWORD` a fost introdusă în MySQL 3.22.
- `ping`

  Verifică dacă serverul MySQL rulează sau nu.
- `processlist`

  Afișează lista firelor de server care se execută la un moment dat.

  Această comandă este similară instrucțiunii SQL `SHOW PROCESSLIST`.
- `refresh`

  Golește zona cache a tabelului și închide și redeschide fișierele jurnal.
- `reload`

  Reîncarcă tabelele de acordare.

  Această comandă este similară instrucțiunii SQL `FLUSH PRIVILEGES`.
- `shutdown`

  Indică serverului să-și înceteze execuția.
- `status`

  Afișează un scurt mesaj de stare de la server.
- `variables`

  Afișează numele și valorile variabilelor serverului.

  Această comandă este similară instrucțiunii SQL `SHOW VARIABLES`.
- `version`

  Regăsește și afișează șirul cu informații privind versiunea serverului.

mysqlbug

Capitolul 2 al manualului de referință MySQL conține o procedură detaliată pentru completarea unui raport de semnalare a unei hibe. Prin respectarea acestei proceduri, vă asigurați că raportul dumneavoastră va fi util și va conține suficiente informații pentru rezolvarea problemei pe care ați semnalat-o. O parte cheie a acestei proceduri constă în utilizarea scriptului `mysqlbug`. Acesta este folosit pentru crearea și expedierea unui raport de semnalare a unei hibe către lista de corespondență MySQL atunci când descoperiți o problemă a sistemului MySQL. Programul `mysqlbug` adună informații despre sistemul dumneavoastră și despre configurația MySQL și apoi vă deschide un editor care include conținutul mesajului poștal care va fi trimis. Editați mesajul pentru a adăuga cât mai multe informații cu putință despre problema pe care o semnalați și apoi scrieți mesajul pentru a vă salva modificările și părăsiți editorul. `mysqlbug` vă va întreba dacă doriți sau nu să trimiteți raportul, apoi îl trimite, dacă dumneavoastră confirmați intenția de a-l trimite.

Vă rugăm să folosiți `mysqlbug` pentru a semnala hibele, dar nu-l utilizați oricum. În multe situații, o „hibă" nu este deloc... o hibă sau este un raport privind un fapt care a fost deja documentat în manualul de referință. De exemplu, de multe ori s-a semnalat ca hibă faptul că instrucțiunea `DELETE FROM` *nume_tabel* raportează un număr al rândurilor afectate egal cu zero. Este adevărat, dar acesta este comportamentul scontat. De asemenea, este documentat în manual ca atare, ca și interogarea pe care trebuie să o folosiți dacă doriți numărul real de rânduri afectate. Arhivele listei de corespondență MySQL reprezintă o altă sursă utilă de informații. Legăturile spre arhive se pot găsi la adresa `http://www.mysql.com/doc.html`.

Utilizare

```
mysqlbug [adresa]
```

Raportul de semnalare a unei hibe este trimis către lista de corespondență MySQL în mod prestabilit. Dacă specificați în linia de comandă o adresă de e-mail, raportul va fi trimis la adresa respectivă. Puteți specifica propria dumneavoastră adresă pentru a testa programul `mysqlbug` fără a trimite raportul spre lista de corespondență; este chiar recomandat să procedați astfel atunci când folosiți scriptul pentru prima dată și poate că nu sunteți sigur cu privire la modul de funcționare al acestuia.

mysqld

mysqld este serverul MySQL. Furnizează accesul programelor client la bazele de date, deci este necesar ca programul să funcționeze, în caz contrar clienții neputând folosi bazele de date administrate de server. Când `mysqld` pornește, deschide porturile de rețea la care va asculta și apoi așteaptă conexiunile client. Programul `mysqld` este multifir și prelucrează fiecare conexiune client folosind un fir separat pentru a asigura concurența între clienți. Interogările care scriu în baza de date sunt executate separat; când serverul începe să execute o asemenea interogare, nu va mai executa nici o altă interogare pentru tabelul implicat decât atunci când interogarea curentă și-a încheiat execuția. Din acest motiv, niciodată doi clienți nu vor putea modifica simultan același rând dintr-un tabel.

Utilizare

```
mysqld [optiuni]
```

Opțiuni standard acceptate de mysqld

`--debug`	`--port`	`--user`
`--help`	`--socket`	`--version`

Rețineți că, deși opțiunea `--socket` este acceptată, opțiunea `-S` nu este, deoarece serverul folosește opțiunea `-Sg` cu o semnificație care nu are nici o legătură cu fișierul soclu.

Dacă este dată opțiunea `--user`, aceasta specifică numele de utilizator pentru contul care se va folosi la rularea serverului. În acest caz, când serverul pornește, caută numele de utilizator și identificatorul de grup ale contului din fișierul parolă și apoi modifică

numele de utilizator și identificatorul de grup proprii astfel încât să fie similare celor ale contului. În acest mod, serverul rulează cu privilegiile asociate utilizatorului respectiv, nu cu privilegiile utilizatorului `root`. (Serverul trebuie să fie pornit sub contul `root` pentru ca opțiunea `--user` să fie eficientă; în caz contrar, nu va putea să-și modifice identificatorul de utilizator.) Această opțiune a fost introdusă în MySQL versiunea 3.22.

Opțiuni specifice serverului `mysqld`

- `-b` *`nume_cale`*, `--basedir=`*`nume_cale`*

  Specifică numele căii de acces spre catalogul de instalare MySQL. Multe alte căi de acces sunt rezolvate începând de la acest catalog, dacă sunt specificate sub formă de căi relative de acces.
- `--big-tables`

  Permite prelucrarea unor seturi de rezultate de mari dimensiuni prin salvarea tuturor rezultatelor temporare pe disc, nu prin stocarea lor în memorie. Astfel, sunt evitate majoritatea mesajelor de tip „tabel plin" care se produc ca rezultat al existenței unei cantități prea mici de memorie pentru stocarea unor seturi de rezultate de mari dimensiuni. Începând cu versiunea MySQL 3.23, această opțiune nu mai este necesară.
- `--bind-address=`*`adresa_ip`*

  Se asociază la adresa IP dată. În mod normal, `mysqld` se asociază la adresa IP prestabilită a gazdei la care rulează serverul. Această opțiune se poate folosi pentru a selecta o adresă alternativă de asociere, dacă o gazdă are mai multe adrese.
- `--bootstrap`

  Această opțiune este folosită de scripturile de instalare la prima instalare a sistemului MySQL.
- `--chroot=`*`nume_cale`*

  Rulează serverul MySQL ancorat la un anumit catalog rădăcină. Opțiunea precizează calea de acces spre catalogul respectiv. Pentru mai multe informații privind rularea într-un mediu `chroot()`, consultați pagina de manual UNIX aferentă comenzii `chroot()`. Această opțiune a fost introdusă în versiunea MySQL 3.22.2.
- `-h` *`nume_cale`*, `--datadir=`*`nume_cale`*

  Specifică numele căii de acces spre catalogul de date MySQL.
- `--default-table-type=`*`tip`*

  Specifică tipul prestabilit de stocare a tabelelor. Valoarea *`tip`* trebuie să fie `isam`, `myisam` sau `heap`. (Valoarea nu este sensibilă la diferența între majuscule și minuscule.) În mod prestabilit, dacă această opțiune nu este specificată, se folosesc tabelele MyISAM. Această opțiune a fost introdusă în versiunea MySQL 3.23.
- `--enable-locking`

  Activează blocarea externă (blocarea sistemului de fișiere) pentru sisteme precum Linux, unde opțiunea prestabilită este `--skip-locking`. Această opțiune a fost introdusă în MySQL 3.22.4.

- `-T [`*`n`*`]`, `--exit-info[=`*`n`*`]`

  Afișează informațiile de depanare la încheierea execuției programului. Această opțiune a fost introdusă în versiunea MySQL 3.22.
- `--init-file=`*`nume_fisier`*

  Specifică un fișier cu instrucțiuni SQL care vor fi executate la pornire.
- `-L`, `--language=`*`nume_limba`*

  Afișează clienților mesaje de eroare în limba specificată. În mod normal, *`nume_limba`* va fi o valoare de genul `english` sau `german`.
- `-l [`*`nume_cale`*`]`, `--log[=`*`nume_cale`*`]`

  Activează caracteristica de consemnare în jurnalul general. Acesta conține informații privind conexiunile clienților și interogările. Dacă parametrul *`nume_cale`* nu este dat, numele fișierului jurnal este *`nume_gazda`*`.log`, unde *`nume_gazda`* este numele gazdei serverului. Dacă *`nume_gazda`* este dat sub formă de cale relativă, aceasta va fi interpretată pornind de la catalogul de date.
- `--log-isam[=`*`nume_fisier`*`]`

  Activează caracteristica de consemnare în fișierele index. Această comandă se folosește numai pentru depanarea tabelelor ISAM/MyISAM.
- `--log-long-format`

  Scrie informații suplimentare în jurnalul de actualizare. Această opțiune a fost introdusă în versiunea MySQL 3.22.7.
- `--log-update[=`*`nume_cale`*`]`

  Activează consemnarea în jurnalul de actualizare. Jurnalul de actualizare conține textul oricărei interogări care modifică tabelele dintr-o bază de date. Dacă nu este dat parametrul *`nume_cale`*, numele fișierului jurnal de actualizare este *`nume_gazda.nnn`*, unde *`nume_gazda`* este numele gazdei serverului, iar *`nnn`* este un număr dintr-o secvență cu o unitate mai mare (numărul, nu secvența – N.T.) decât cel al jurnalului de actualizare anterior.

  Dacă *`nume_cale`* este dat și componenta finală nu conține extensii, serverul adaugă o extensie numerică de forma *`nnn`*, având o valoare conform celor specificate anterior. În cazul în care componenta finală a numelui de cale include o extensie, numele respectiv va fi folosit, fără modificări, ca nume de fișier pentru jurnalul de actualizare. Dacă *`nume_cale`* este specificat sub formă de cale relativă, este interpretat pornind de la catalogul de date.
- `--low-priority-updates`

  Acordă operațiilor de scriere o prioritate mai redusă decât operațiilor de regăsire. Această opțiune a fost introdusă în versiunea MySQL 3.23. (De la versiunea MySQL 3.22.5 și până la versiunea 3.23, opțiunea a purtat numele `--low-priority-inserts`.)
- `-n --new`

  Folosește rutine noi, posibil nesigure. Acestea sunt caracteristici ale sistemului MySQL care nu au fost încă declarate stabile. Utilizați această opțiune numai dacă aveți spirit de aventură.

- `-o, --old-protocol`

  Folosește protocolul care a fost utilizat pentru comunicațiile client-server anterior versiunii MySQL 3.21. Această opțiune poate fi necesară dacă serverul comunică cu programe client foarte vechi.
- `--one-thread`

  Rulează folosind un singur fir; folosit pentru depanare sub Linux, care în mod normal folosește minimum trei fire. Această opțiune a fost introdusă în versiunea MySQL 3.22.2.
- `--pid-file=`*`nume_cale`*

  Când programul `mysqld` pornește, își scrie identificatorul de proces (PID) într-un fișier. Această opțiune specifică numele căii de acces a fișierului PID. Fișierul poate fi folosit de alte procese pentru a determina numărul de proces al serverului, în mod caracteristic pentru trimiterea unui semnal. De exemplu, `mysql.server` citește fișierul atunci când trimite serverului un semnal de închidere.
- `--safe-mode`

  Această opțiune este similară opțiunii `--skip-new`, dar dezactivează un număr mai mare de facilități. O puteți încerca dacă MySQL pare instabil sau dacă anumite interogări complexe tind să determine rezultate incorecte. Dacă utilizarea acestei opțiuni îmbunătățește funcționarea serverului, vă rugăm să notați acest fapt atunci când folosiți programul `mysqlbug` pentru a raporta problemele pe care le întâlniți.
- `-Sg, --skip-grant-tables`

  Nu folosește tabelele de acordare pentru a verifica conexiunile clienților. Astfel, toți clienții primesc drepturi de acces total, pentru a executa orice operații. Puteți cere serverului să înceapă a folosi tabelele de acordare emițând o comandă `mysqladmin flush-privileges`.
- `--skip-host-cache`

  Dezactivează utilizarea zonei cache cu numele gazdei.
- `--skip-locking`

  Nu permite utilizarea blocării de sistem (blocarea sistemului de fișiere). Dacă folosiți această opțiune, trebuie să opriți serverul înainte de a folosi `isamchk` sau `myisamchk` pentru a repara tabelele; în caz contrar, riscați să le deteriorați.
- `--skip-name-resolve`

  Dacă este specificată această opțiune, tabelele de acordare trebuie să specifice gazdele în funcție de numărul IP sau folosind expresia `localhost`.
- `--skip-networking`

  Nu permite stabilirea conexiunilor TCP/IP. Clienții trebuie să se conecteze de la gazda locală, folosind `localhost`.
- `--skip-new`

  Nu folosește rutinele noi, posibil nesigure. Vezi opțiunea `--new`.

- `--skip-show-database`

  Nu permite utilizatorilor fără privilegii să emită interogări `SHOW DATABASES` sau să folosească instrucțiunea `SHOW TABLES` în baze de date pentru care nu au acces. Această opțiune a fost introdusă în MySQL versiunea 3.23.
- `--skip-thread-priority`

  În mod normal, interogările care modifică tabele rulează la o prioritate mai ridicată decât cele care regăsesc date. Dacă nu se dorește acest lucru, această opțiune determină serverul să nu acorde priorități diferite pentru diverse tipuri de interogări.
- `-t `*`nume_cale`*`, --tmpdir=`*`nume_cale`*

  Specifică numele căii de acces a catalogului care se va folosi pentru fișierele temporare. Această opțiune a fost introdusă în versiunea MySQL 3.22.4.

Opțiunile specifice sistemului Windows sunt următoarele:

- `--console`

  Afișează o fereastră a consolei pentru mesaje de eroare. Această opțiune a fost introdusă în versiunea MySQL 3.22.4.
- `--flush`

  Evacuează toate tabelele pe disc, după fiecare actualizare. Aceasta reduce riscul alterării tabelelor în eventualitatea unei căderi a sistemului, dar provoacă serioase degradări ale performanțelor. Această opțiune a fost introdusă în versiunea MySQL 3.22.9.
- `--install`

  Instalează `mysqld-nt` ca serviciu (numai pentru Windows NT).
- `--remove`

  Elimină `mysqld-nt` ca serviciu (numai pentru Windows NT).
- `--standalone`

  Pornește `mysqld_nt` ca program autonom, nu ca serviciu (numai pentru Windows NT).

Variabile pentru mysqld

Pentru a vedea care sunt valorile pe care `mysqld` le va utiliza în mod prestabilit, folosiți această comandă:

```
% mysqld --help
```

Pentru a vedea care sunt valorile pe care le folosește instanța serverului `mysqld` care se execută la un moment dat, utilizați această comandă:

```
% mysqladmin variables
```

De asemenea, puteți verifica valorile curente ale variabilelor emițând o interogare `SHOW VARIABLES`. Variabilele `mysqld` care pot fi configurate cu opțiunea `--set-variable` sunt prezentate în următoarea listă:

- `back_log`

  Numărul de cereri de conexiune suspendate care pot fi așezate la coadă în timp ce sunt prelucrate conexiunile actuale.

- `connect_timeout`

  Pe durata protocolului inițial de conexiune, `mysqld` va aștepta pachetele timp de `connect_timeout` secunde.

- `delayed_insert_timeout`

  Când rutina de tratare a operațiilor `INSERT DELAYED` termină de inserat rândurile așezate la coadă, așteaptă un număr de secunde egal cu valoarea acestui parametru, pentru a vedea dacă mai sosesc rânduri `INSERT DELAYED`. Dacă mai sosesc, le manipulează; dacă nu, își încheie execuția.

- `delayed_insert_limit`

  Numărul de rânduri din instrucțiunile `DELAYED INSERT` care vor fi inserate într-un tabel înainte de a verifica dacă au mai sosit alte noi instrucțiuni `SELECT` pentru tabelul respectiv. Dacă acestea au sosit, operația de inserție este suspendată pentru a permite executarea regăsirilor.

- `delayed_queue_size`

  Numărul de rânduri care pot fi așezate la coadă pentru instrucțiunile `INSERT DELAYED`. În cazul în care coada este plină, celelalte instrucțiuni `INSERT DELAYED` sunt blocate până când se creează spațiu în coadă.

- `flush_time`

  Dacă această variabilă are o valoare diferită de zero, tabelele sunt închise pentru evacuarea pe disc a modificărilor suspendate la fiecare `flush_time` secunde. Valoarea prestabilită pentru Windows este de 30 de minute; pentru a o modifica, folosiți opțiunea `--flush`.

- `join_buffer_size`

  Dimensiunea bufferului folosit la uniri complete (adică bufferul pentru uniri executate fără utilizarea indexurilor).

  Anterior versiunii MySQL 3.23, numele acestei variabile era `join_buffer`.

- `key_buffer_size`

  Dimensiunea bufferului folosit pentru blocurile de index. Acest buffer este partajat de firele rutinelor de tratare a conexiunilor.

  Anterior versiunii MySQL 3.23, numele acestei variabile era `key_buffer`.

- `long_query_time`

  Intervalul de timp, în secunde, care definește o interogare „lentă". Orice interogare care necesită un interval de timp mai mare decât acesta determină incrementarea contorului `Slow_queries`.

- `max_allowed_packet`

  Dimensiunea maximă a bufferului folosit pentru comunicarea între server și client. Inițial, dimensiunea alocată a bufferului este de `net_buffer_length` octeți, dar poate crește până la `max_allowed_packet` octeți, dacă este necesar.

- `max_connections`

  Numărul maxim de conexiuni client simultane permise.

- `max_connect_errors`

  Numărul de conexiuni ratate de la o gazdă, permise anterior blocării oricăror tentative de conectare de la gazda respectivă. Această operație se bazează pe faptul că este posibil să se încerce o pătrundere ilegală în rețea de la gazda respectivă. Pentru evacuarea zonei cache aferentă gazdelor în vederea reactivării gazdelor blocate se poate folosi instrucțiunea `FLUSH HOSTS` sau comanda `mysqladmin flush-hosts`.

- `max_delayed_insert_threads`

  Numărul maxim de rutine de tratare care vor fi inițiate pentru tratarea instrucțiunilor `INSERT DELAYED`. Toate instrucțiunile de acest gen care sunt primite în timp ce numărul maxim de rutine de tratare se află în uz vor fi tratate ca instrucțiuni ce nu sunt de tip `DELAYED`.

- `max_join_size`

  Utilitarul de optimizare MySQL estimează numărul de rânduri pe care îl va returna o operație de unire. Dacă valoarea estimată depășește `max_join_size` rânduri, va fi returnată o eroare. Această variabilă se poate folosi dacă utilizatorii au tendința de a scrie interogări `SELECT` la întâmplare, care returnează un număr aleator de rânduri.

- `max_sort_length`

  Valorile de tip `BLOB` sau `TEXT` sunt sortate folosind primii `max_sort_length` octeți ai fiecărei valori.

- `net_buffer_length`

  Dimensiunea inițială a bufferului folosit pentru comunicarea între server și client. Acest buffer poate fi extins până la un maximum de `max_allowed_packets` octeți.

- `record_buffer`

  Dimensiunea bufferului folosit de fire care execută baleieri secvențiale ale tabelelor.

- `sort_buffer`

  Dimensiunea bufferului folosit de fire pentru execuția operațiilor de sortare (`GROUP BY` sau `ORDER BY`). Această valoare corespunde unui singur fir. În mod normal, dacă puteți avea mai multe fire care execută sortarea simultan, nu este indicat ca această valoare să fie foarte mare (peste 1MB).

- `table_cache`

  Numărul maxim de tabele care pot fi deschise. Această zonă cache este partajată între fire.

- `tmp_table_size`

  Numărul maxim de octeți permis pentru tabelele temporare. Dacă un tabel depășește această dimensiune, `mysqld` returnează clientului o eroare.

- `thread_stack`

  Dimensiunea stivei pentru fiecare fir.

- `wait_timeout`

  Intervalul de timp (în secunde) în care o conexiune poate rămâne inactivă înainte ca serverul să o închidă.

mysqldump

Programul `mysqldump` scrie conținutul tabelelor din bazele de date în fișiere text. Aceste fișiere se pot folosi pentru o diversitate de scopuri, precum executarea copiilor de siguranță a bazelor de date, mutarea bazelor de date într-un alt server sau configurarea unei baze de date de test, în funcție de conținutul unei baze de date existente.

În mod prestabilit, datele de ieșire pentru fiecare tabel transferat sunt alcătuite dintr-o instrucțiune `CREATE TABLE` care re-creează tabelul, urmată de un set de instrucțiuni `INSERT` care reîncarcă datele incluse în tabel. Dacă este specificată opțiunea `--tab`, conținutul tabelului este scris într-un fișier de date în formatul brut, iar instrucțiunea SQL de creare a unui tabel este scrisă într-un fișier separat.

Utilizare

```
mysqldump [optiuni] nume_db [nume_tabel] ...
```

Dacă nu sunt menționate nume de tabele după numele bazei de date *nume_db*, vor fi transferate toate tabelele din baza de date. În caz contrar, vor fi transferate numai tabelele denumite.

Modalitatea cea mai frecventă de utilizare a programului `mysqldump` este următoarea:

```
% mysqldump --opt nume_db > nume_fisier_copie
```

Opțiuni standard acceptate de programul mysqldump

--compress	--host	--port	--user
--debug	--password	--set-variable	--verbose
--help	--pipe	--socket	--version

Opțiuni specifice programului mysqldump

Următoarele opțiuni controlează modul de operare a programului `mysqldump`. Secțiunea următoare, „Opțiuni privind formatul datelor", descrie opțiunile care se pot folosi în conjuncție cu opțiunea `--tab` pentru a indica formatul fișierelor de date.

- `--add-drop-table`

  Adaugă o instrucțiune `DROP TABLE IF EXISTS` înainte de fiecare instrucțiune `CREATE TABLE`.

- `--add-locks`

  Adaugă instrucțiunile `LOCK TABLE` și `UNLOCK TABLE` în jurul setului de instrucțiuni `INSERT` care încarcă datele pentru fiecare tabel.

- `--allow-keywords`

  Permite crearea numelor de coloană care sunt cuvinte cheie.

- `-c, --complete-insert`

  Folosește instrucțiuni `INSERT` care denumesc fiecare coloană care va fi inserată.

- `--delayed-insert`

  Scrie instrucțiuni `INSERT DELAYED`.

- `-e, --extended-insert`

  Scrie instrucțiuni `INSERT` pe mai multe rânduri.

- `-F, --flush-logs`

  Golește fișierele jurnal ale serverului înainte de a începe transferul.

- `-f, --force`

  Continuă execuția chiar dacă se produc erori.

- `--full`

  Adaugă informații suplimentare la instrucțiunile `CREATE TABLE` pe care le generează programul `mysqldump`, precum tipul tabelului, valoarea `AUTO_INCREMENT` inițială și așa mai departe. Acestea sunt informațiile pe care le puteți specifica în partea *optiuni_tabel* din sintaxa instrucțiunii `CREATE TABLE` (vezi Anexa D).

- `-l, --lock-tables`

  Obține o blocare pentru toate tabelele transferate, înainte de a le transfera.

- `-t, --no-create-info`

  Nu scrie instrucțiunile `CREATE TABLE`.

- `-d, --no-data`

  Nu scrie datele din tabel.

- `--opt`

  Optimizează viteza de transfer a tabelelor și scrie un fișier de transfer care este optim pentru viteza de reîncărcare. Această opțiune activează opțiunile `--add-drop-table`, `--add-locks`, `--all`, `--extended-insert`, `--quick` și `--lock-tables`.

- `-q, --quick`

  În mod prestabilit, `mysqldump` citește în memorie întreg conținutul unui tabel și apoi îl scrie. Această opțiune determină scrierea la ieșire a fiecărui rând imediat ce acesta a fost citit de la server.

  Dacă folosiți această opțiune, nu trebuie să suspendați programul `mysqldump`; astfel, veți determina serverul să aștepte, ceea ce afectează funcționarea altor clienți.

- `-T dir_transfer, --tab=dir_transfer`

  Această opțiune determină programul `mysqldump` să scrie două fișiere pentru fiecare tabel, folosind *dir_transfer* ca locație a fișierelor. Catalogul respectiv trebuie să existe deja. Pentru fiecare tabel *nume_tabel*, este scris un fișier *dir_transfer*/*nume_tabel*`.txt`, care conține datele din tabel, precum și un fișier *dir_transfer*/*nume_tabel*`.sql`, care conține instrucțiunea `CREATE TABLE` pentru tabel. Pentru a folosi această opțiune, trebuie să dispuneți de privilegiul `FILE`.

Pentru a evita confuzia, cel mai bine este să rulați programul `mysqldump` pe gazda serverului atunci când folosiți această opțiune. În caz contrar, unele dintre fișiere sunt scrise la server și unele la client. *dir_transfer* este folosit pe gazda serverului pentru fișierele `*.sql`, respectiv pe gazda clientului pentru fișierele `*.txt`. Fișierele `*.sql` vor fi în posesia contului folosit pentru rularea serverului, iar fișierele `*.txt` vor fi în posesia dumneavoastră.

În mod prestabilit, fișierul de date conține linii care se termină prin caractere linie nouă și care sunt alcătuite din valori ale coloanelor separate prin tabulatori. Acest format se poate modifica folosind opțiunile descrise în secțiunea „Opțiuni privind formatul datelor".

- `-w` *clauza_unde*, `--where=`*clauza_unde*

  Transferă numai înregistrările selectate de clauza `WHERE` dată de *clauza_unde*. Trebuie să delimitați clauza între ghilimele, pentru a împiedica shell-ul să o interpreteze sub formă de mai multe argumente în linia de comandă. Această opțiune a fost introdusă în versiunea MySQL 3.22.7.

Opțiuni privind formatul datelor

Dacă specificați opțiunea `--tab` sau `-T` în vederea generării unui fișier de date separat pentru fiecare tabel, se aplică mai multe opțiuni suplimentare. Poate că va fi necesar să delimitați valoarea opțiunii folosind caracterele de citare adecvate. Aceste opțiuni sunt analoge opțiunilor privind formatul datelor specifice instrucțiunii `LOAD DATA`. Consultați informațiile aferente instrucțiunii `LOAD DATA` în Anexa D.

- `--fields-enclosed-by=`*caracter*

  Specifică faptul că valorile din coloane trebuie să fie delimitate folosind caracterul respectiv, de regulă un caracter de citare. În mod prestabilit, nu se folosesc nici un fel de semne de citare.

- `--fields-escaped-by=`*caracter*

  Precizează caracterul escape pentru modificarea semnificației caracterelor speciale. În mod prestabilit, nu se folosește un asemenea caracter.

- `--fields-optionally-enclosed-by=`*caracter*

  Specifică faptul că valorile din coloane trebuie delimitate folosindu-se caracterul dat, de obicei un caracter de citare. Caracterul este folosit pentru coloane care nu conțin valori numerice. În mod prestabilit, nu se folosește un asemenea caracter.

- `--fields-terminated-by=`*caracter*

  Precizează caracterul de separare între valorile coloanelor utilizat în fișierele de date. În mod prestabilit, valorile sunt separate prin caractere de tip linie nouă.

- `--lines-terminated-by=`*sir*

  Specifică șirul (pot fi mai multe caractere) de scris la sfârșitul liniilor care conțin datele de ieșire. În mod prestabilit, sunt scrise caractere linie nouă.

Variabile pentru programul `mysqldump`

Următoarele variabile `mysqldump` pot fi configurate cu ajutorul opțiunii `--set-variable`:

- `max_allowed_packet`

  Dimensiunea maximă a bufferului folosit pentru comunicațiile între server și client.

- `net_buffer_length`

  Dimensiunea inițială a bufferului folosit pentru comunicațiile între server și client. Acest buffer poate fi extins până la valoarea maximă de `max_allowed_packet` octeți.

`mysqlimport`

Utilitarul `mysqlimport` este un utilitar de încărcare de mare capacitate pentru citirea conținutului fișierelor text în tabelele existente. Funcționează ca interfață în linie de comandă pentru instrucțiunea SQL `LOAD DATA` și constituie o modalitate eficientă de introducere a rândurilor în tabele.

Utilizare

```
mysqlimport [optiuni] nume_db nume_fisier ...
```

Argumentul *nume_db* specifică baza de date care conține tabelele unde doriți să încărcați date. Tabelele de încărcat sunt determinate din argumentele nume de fișier. Pentru fiecare nume de fișier, orice extensie care începe de la primul punct inclus în numele de fișier este eliminată, iar numele de bază rămas este folosit ca nume al tabelului în care trebuie încărcat fișierul. De exemplu, `mysqlimport` va încărca conținutul fișierului `presedinte.txt` în tabelul `presedinte`.

Opțiuni standard acceptate de `mysqlimport`

`--compress`	`--host`	`--port`	`--user`
`--debug`	`--password`	`--silent`	`--verbose`
`--help`	`--pipe`	`--socket`	`--version`

Opțiuni specifice programului `mysqlimport`

Următoarele opțiuni controlează modul în care programul `mysqlimport` prelucrează fișierele de intrare. Următoarea secțiune, „Opțiuni privind formatul datelor", descrie opțiuni care se pot folosi pentru a indica formatul datelor din fișierele de intrare.

- `-d`, `--delete`

  Golește fiecare tabel înainte de a încărca date în acesta.

- `-f`, `--force`

  Continuă încărcarea rândurilor, chiar dacă se produc erori.

- `-i`, `--ignore`

  Când un rând de intrare conține o valoare a unei chei unice deja existentă în tabel păstrează rândul existent și ignoră rândul de intrare.

- `-L, --local`

  În mod prestabilit, programul `mysqlimport` permite serverului să citească fişierul de date, ceea ce înseamnă că fişierul trebuie să se găsească pe gazda serverului şi că dumneavoastră trebuie să dispuneţi de privilegiul `FILE`. Dacă specificaţi opţiunea `--local`, indicaţi programului `mysqlimport` să citească personal fişierul de date şi să-l trimită serverului. Acest procedeu este mai lent, dar funcţionează atunci când rulaţi `mysqlimport` pe un alt calculator decât gazda serverului, dar şi când rulaţi programul respectiv pe gazda serverului, chiar dacă nu aveţi privilegiul `FILE`.

  Opţiunea `--local` a fost introdusă în versiunea MySQL 3.22.15.
- `-l, --lock-tables`

  Blochează fiecare tabel înainte de a introduce date în acesta.
- `-r, --replace`

  Când un rând de intrare conţine o valoare a unei chei unice deja existentă în tabel, înlocuieşte rândul existent cu rândul de intrare.

Opţiuni privind formatul datelor

În mod prestabilit, `mysqlimport` presupune că fişierele de date conţin linii terminate prin caractere linie nouă şi alcătuite din valori separate prin tabulatori. Formatul aşteptat poate fi modificat folosind următoarele opţiuni. Poate că este necesar să delimitaţi valoarea opţiunii între caracterele de citare adecvate. Aceste opţiuni sunt analoge opţiunilor privind formatul datelor caracteristice instrucţiunii `LOAD DATA`. Consultaţi informaţiile aferente instrucţiunii `LOAD DATA` în Anexa D.

- `--fields-enclosed-by=`*`caracter`*

  Specifică faptul că valorile din coloane trebuie să fie delimitate folosind caracterul respectiv, de regulă un caracter de citare. În mod prestabilit, se presupune că valorile nu sunt delimitate prin nici un fel de semne de citare.
- `--fields-escaped-by=`*`caracter`*

  Precizează caracterul escape folosit pentru modificarea semnificaţiei caracterelor speciale. În mod prestabilit, nu există un asemenea caracter.
- `--fields-optionally-enclosed-by=`*`caracter`*

  Specifică faptul că valorile din coloane trebuie delimitate folosindu-se caracterul dat, de obicei un caracter de citare.
- `--fields-terminated-by=`*`caracter`*

  Precizează caracterul de separare între valorile coloanelor. În mod prestabilit, valorile sunt separate prin caractere de tip linie nouă.
- `--lines-terminated-by=`*`sir`*

  Specifică şirul (pot fi mai multe caractere) situat la sfârşitul liniilor care conţin datele de intrare. În mod prestabilit, se presupune că liniile respective se termină prin caractere linie nouă.

mysqlshow

Utilitarul `mysqlshow` afişează bazele de date, tabelele dintr-o bază de date, respectiv informaţii despre coloanele sau indexurile dintr-un tabel. Se comportă ca o interfaţă în linie de comandă pentru instrucţiunea SQL `SHOW`.

Utilizare

```
mysqlshow [optiuni] [nume_db [nume_tabel [nume_coloana]]]
```

Dacă nu este specificat numele unei baze de date, `mysqlshow` afişează toate bazele de date din gazda serverului. Dacă este dat numele unei baze de date, dar nu este specificat numele unui tabel, vor fi afişate toate tabelele din baza de date. Dacă sunt specificate numele bazei de date şi numele tabelului, dar nu este specificat numele unei coloane, afişează coloanele din tabel. Dacă sunt specificate toate numele, utilitarul `mysqlshow` afişează informaţii despre coloana indicată.

Dacă argumentul final conţine un caracter de înlocuire pentru interpretor (`*` sau `?`), datele de ieşire sunt limitate la valorile care corespund modelului creat prin utilizarea caracterului de înlocuire. Cele două caractere menţionate mai sus sunt tratate ca şi caracterele de înlocuire SQL `%` şi `_` aferente operatorului `LIKE`.

Opţiuni standard acceptate de mysqlshow

`--compress`	`--host`	`--port`	`--version`
`--debug`	`--password`	`--sockett`	
`--help`	`--pipe`	`--user`	

Opţiuni specifice programului mysqlshow

- `-i, --status`

  Afişează aceleaşi categorii de informaţii despre tabel ca şi cele afişate de instrucţiunea `SHOW TABLE STATUS`.

  Opţiunea `--status` a fost introdusă în versiunea MySQL 3.23.
- `-k, --keys`

  Afişează informaţii despre indexurile din tabel, în afară de informaţiile despre coloanele din tabele. Această opţiune are sens numai dacă specificaţi un nume de tabel.

safe_mysqld

`safe_mysqld` este un script care porneşte serverul `mysqld` şi îl monitorizează. Dacă serverul îşi încheie accidental execuţia, scriptul `safe_mysqld` îl reporneşte.

Utilizare

Scriptul `safe_mysqld` acceptă aceleaşi opţiuni ca şi `mysqld`.

ANEXA F

Referință API C

Această anexă descrie interfața de programare a aplicațiilor în limbajul C pentru biblioteca client MySQL. Interfața API constă dintr-un set de funcții pentru comunicarea cu serverele MySQL și obținerea accesului la bazele de date, precum și un set de tipuri de date folosite de aceste funcții.

Această anexă servește drept referință, deci include numai scurte fragmente de programe care ilustrează utilizarea bibliotecilor client. Pentru programele client complete și note cu privire la redactarea lor, examinați capitolul 6, „Interfața API MySQL pentru C".

Compilare și legare

La nivelul sursă, biblioteca client este definită în fișierul antet `mysql.h`, care poate fi inclus în propriile dumneavoastră fișiere sursă:

```
#include mysql.h
```

Pentru a arăta compilatorului unde să găsească acest fișier, poate că este necesar să specificați o opțiune `-I/cale/spre/dir/include`. De exemplu, dacă fișierele dumneavoastră antet MySQL sunt instalate în catalogul `usr/include/mysql` sau `/usr/local/mysql/include`, puteți compila fișierele sursă folosind comenzi care se prezintă astfel:

```
% gcc -I/usr/include/mysql -c progmeu.c
% gcc -I/usr/local/mysql/include -c progmeu.c
```

Dacă trebuie să folosiți alte fișiere antet MySQL, acestea sunt localizate în același catalog ca și `mysql.h`. De exemplu, `mysql_com.h` conține constante și macroinstrucțiuni pentru interpretarea metadatelor aferente rezultatelor interogării. (`mysql_com.h` este inclus în `mysql.h`, deci, deși puteți dori să examinați acest fișier pentru a vedea care este conținutul acestuia, aveți acces la fișierul respectiv prin simpla includere a fișierului `mysql.h`.) Fișierele antet `errmsg.h` și `mysqld_error.h` conțin constante pentru coduri de eroare.

La nivelul obiect, biblioteca client este furnizată sub forma bibliotecii `mysqlclient`. Pentru a lega această bibliotecă în programul dumneavoastră, specificați opțiunea `-lmysqlclient` în comanda de legare. Probabil că va trebui să indicați programului de legare unde să găsească biblioteca folosind o opțiune `-L/cale/spre/cat/bibl`. De exemplu:

```
% gcc -o progmeu progmeu.o -L/usr/lib/mysql -lmysqlclient
% gcc -o progmeu progmeu.o -L/usr/local/mysql/lib -lmysqlclient
```


Tipuri de date din interfața API C

Tipurile de date pentru biblioteca client MySQL sunt proiectate astfel încât să reprezinte entitățile pe care le utilizați în cursul unei sesiuni cu serverul. Există tipuri pentru conexiunea în sine, pentru rezultatele unei interogări, pentru un rând din cadrul unui rezultat, precum și pentru metadate (informații descriptive despre coloanele care compun un rezultat). În expunerea următoare, noțiunile de „coloană" și „câmp" sunt sinonime.

Tipuri de date scalare

Tipurile de date scalare din MySQL reprezintă valori precum întregi de foarte mari dimensiuni, valori booleene și deplasamente de câmp.

- `my_ulonglong`

  Un tip de întreg lung, folosit pentru valorile returnate de funcții precum `mysql_affected_rows()`, `mysql_num_rows()` și `mysql_insert_id()`, care returnează numere de rânduri sau alte numere de dimensiuni mari.

  Pentru a afișa o valoare `my_ulonglong`, convertiți-o la `unsigned long` și folosiți formatul `%lu`. Dacă nu procedați astfel, valoarea nu va fi afișată corect pe unele sisteme.

- `my_bool`

  Un tip boolean, folosit pentru valorile returnate de funcțiile `mysql_change_user()` și `mysql_eof()`.

- `MYSQL_FIELD_OFFSET`

  Acest tip de date este folosit de funcțiile `mysql_field_tell()` și `mysql_field_seek()` pentru reprezentarea deplasamentelor din setul de structuri `MYSQL_FIELD` aferente setului de rezultate curent.

Tipuri de date non-scalare

Tipurile non-scalare folosite în MySQL reprezintă structuri sau tablouri. Fiecare structură `MYSQL` și `MYSQL_RES` trebuie asimilată cu o „cutie neagră". Cu alte cuvinte, trebuie să faceți referire numai la structura în sine, nu la elementele din structură. Tipurile `MYSQL_ROW` și `MYSQL_FIELD` nu prezintă o asemenea restricție; acestea sunt accesibile direct pentru obținerea datelor și a metadatelor returnate ca rezultat al unei interogări.

- `MYSQL`

  Principalul tip de bibliotecă client este structura `MYSQL`, care este folosită pentru rutinele de tratare a conexiunilor. O rutină de tratare conține informații cu privire la starea unei conexiuni din cadrul unui server. Dumneavoastră inițializați o structură `MYSQL` și apoi o transferați unei rutine de conexiune pentru a deschide o sesiune server. După ce ați stabilit conexiunea, puteți folosi rutina de tratare pentru emiterea interogărilor, pentru a genera seturi de rezultate, pentru a obține informații referitoare la erori și așa mai departe.

- `MYSQL_RES`

  Interogări precum `SELECT` sau `SHOW` returnează datele la client prin intermediul unui set de rezultate, reprezentat sub forma unei structuri `MYSQL_RES`. Această structură conține informații cu privire la rândurile returnate de interogare.

După ce ați obținut un set de rezultate, puteți apela funcții API pentru a obține datele din setul de rezultate (valorile datelor din fiecare rând al setului) sau metadate (informații despre rezultat, precum numărul de coloane, tipurile și lungimile acestora etc.)

- MYSQL_ROW

Tipul MYSQL_ROW conține valorile cuprinse într-un rând de date, reprezentat sub formă de tablou din șiruri de octeți numărate. Toate valorile sunt returnate sub formă de șir (chiar și numerele), cu excepția faptului că, dacă o valoare dintr-un rând este NULL, este reprezentată în structura MYSQL_ROW sub forma unui pointer C NULL.

Numărul de valori dintr-un rând este dat de mysql_num_fields(). Valoarea coloanei a i-a dintr-un rând este dată de row[i], unde i variază de la 0 la mysql_num_fields(res_set)-1. (res_set este un pointer spre un set de rezultate din structura MYSQL_RES.)

Rețineți că tipul MYSQL_ROW este deja un pointer, deci veți declara o variabilă de tip rând astfel:

```
MYSQL_ROW rand;      /* corect */
```

Declararea nu se va face astfel:

```
MYSQL_ROW *rand;     /* incorect */
```

Valorile dintr-un tip MYSQL_ROW se termină cu nul, deci valorile care nu sunt de tip binar pot fi tratate sub formă de șiruri care se termină cu nul. Totuși, valorile de date care pot conține date binare ar putea include intern octeți nuli și trebuie să fie tratate ca șiruri numărate. Pentru a obține lungimea valorilor din rând, apelați funcția mysql_fetch_lengths() astfel:

```
unsigned long *lungime;
lungime = mysql_fetch_lengths(res_set);
```

Lungimea returnată de mysql_fetch_lengths() pentru valorile NULL este 0.

- MYSQL_FIELD

Biblioteca client folosește structurile MYSQL_FIELD pentru a reprezenta metadate cu privire la coloanele din setul de rezultate – o structură pentru fiecare coloană. Numărul structurilor MYSQL_FIELD este dat de mysql_num_fields(). Puteți obține accesul la structuri succesive de câmpuri apelând funcția mysql_fetch_field(), respectiv vă puteți deplasa înainte și înapoi printre structuri cu mysql_field_tell() și mysql_field_seek().

Structura MYSQL_FIELD este utilă pentru prezentarea sau interpretarea conținutului rândurilor de date. Structura se prezintă astfel:

```
typedef struct st_mysql_field {
  char *name;
  char *table;
  char *def;
  enum enum_field_types type;
  unsigned int length;
  unsigned int max_length;
  unsigned int flags;
  unsigned int decimals;
} MYSQL_FIELD;
```


Membrii unei structuri MYSQL_FIELD au următoarele semnificații:

- name

  Numele coloanei, ca șir terminat cu nul. Dacă valorile coloanei sunt calculate ca rezultat al unei expresii, nume reprezintă expresia respectivă, sub formă de șir. Dacă o coloană sau o expresie primește un alias de coloană, nume este numele aliasului. De exemplu, interogarea următoare are ca rezultat valori pentru nume precum col_mea, 4*(col_mea+1) și mc:

  ```
  SELECT col_mea, 4*(col_mea+1), col_mea AS mc ...
  ```

- table

  Numele tabelului din care provine coloana, ca șir terminat cu nul. Dacă valorile din coloană sunt calculate ca rezultat al unei expresii, table este un șir vid. De exemplu, dacă emiteți o interogare precum cea indicată mai jos, numele tabelului pentru prima coloană (col_mea) este tabel_meu, în timp ce numele tabelului pentru a doua coloană (col_mea+0) este șirul vid:

  ```
  SELECT col_mea, my_col+0 FROM tabel_meu ...
  ```

- def

  Valoarea prestabilită pentru coloană, ca șir terminat cu nul. Acest membru al structurii MYSQL_FIELD este determinat numai dacă apelați funcția mysql_list_fields(), în caz contrar având valoarea NULL. (Valorile prestabilite ale coloanelor pot fi de asemenea obținute prin emiterea unei interogări DESCRIBE *nume_tabel* sau SHOW FIELDS FROM *nume_tabel* și prin examinarea setului de rezultate.)

- type

  Tipul coloanei. Tipul unei coloane calculate ca rezultat al unei expresii este determinat din tipurile elementelor expresiei. De exemplu, în cazul în care col_mea este o coloană de tip VARCHAR(20), type are valoarea FIELD_TYPE_VAR_STRING, în timp ce atributul type al valorii LENGTH(col_mea) este FIELD_TYPE_LONGLONG.

Valorile posibile ale membrului type sunt enumerate în mysql_com.h și prezentate în tabelul F.1.

Tabelul F.1 Valorile membrului type al structurii MYSQL_FIELD

Valoarea membrului type	Tipul coloanei
FIELD_TYPE_BLOB	BLOB sau TEXT
FIELD_TYPE_DATE	DATE
FIELD_TYPE_DATETIME	DATETIME
FIELD_TYPE_DECIMAL	DECIMAL sau NUMERIC
FIELD_TYPE_DOUBLE	DOUBLE sau REAL
FIELD_TYPE_ENUM	ENUM

Continuare

Tabelul F.1 Continuare

Valoarea membrului `type`	Tipul coloanei
FIELD_TYPE_FLOAT	FLOAT
FIELD_TYPE_INT24	MEDIUMINT
FIELD_TYPE_LONG	INT
FIELD_TYPE_LONGLONG	BIGINT
FIELD_TYPE_NULL	NULL
FIELD_TYPE_SET	SET
FIELD_TYPE_SHORT	SMALLINT
FIELD_TYPE_STRING	CHAR
FIELD_TYPE_TIME	TIME
FIELD_TYPE_TIMESTAMP	TIMESTAMP
FIELD_TYPE_TINY	TINYINT
FIELD_TYPE_VAR_STRING	VARCHAR
FIELD_TYPE_YEAR	YEAR

Puteți vedea referințe la `FIELD_TYPE_CHAR` în fișiere sursă mai vechi; acesta a fost un tip de un octet, care acum se numește `FIELD_TYPE_TINY`.

- `length`

  Lungimea coloanei, conform specificațiilor din instrucțiunea `CREATE TABLE` folosită pentru crearea tabelelor. Lungimea unei coloane calculate ca rezultat al unei expresii este determinată din elementele expresiei.

- `max_length`

  Lungimea valorii celei mai mari existente efectiv în setul de rezultate. De exemplu, dacă o coloană din setul de rezultate conține valorile „Nelu", „Paul" și „Constantin", valoarea `max_length` pentru coloană este 10.

  Deoarece valoarea `max_length` poate fi determinată numai după examinarea tuturor rândurilor, această valoare este semnificativă numai pentru seturile de rezultate create cu `mysql_store_result()`. `max_length` este egală cu 0 pentru seturile de rezultate create cu `mysql_use_result()`.

- `flags`

  Membrul `flags` specifică atributele pentru coloane. Atributele sunt reprezentate prin biți individuali în cadrul valorii `flags`. Biții pot fi testați prin intermediul constantelor de tip mască de biți prezentate în tabelul F.2. De exemplu, pentru a determina dacă valorile unei coloane sunt sau nu de tipul `UNSIGNED`, testați valoarea `flags` astfel:

```
if (field-flags & UNSIGNED_FLAG)
  printf ("%s valori sunt UNSIGNED\n", field-name);
```


Tabelul F.2 Valorile membrului flags al structurii `MYSQL_FIELD`

Valoarea `flags`	Semnificație
AUTO_INCREMENT_FLAG	Coloana are atributul AUTO_INCREMENT
BINARY_FLAG	Coloana are atributul BINARY
MULTIPLE_KEY_FLAG	Coloana face parte dintr-un index NON-UNIQUE
NOT_NULL_FLAG	Coloana nu poate conține valori NULL
PRI_KEY_FLAG	Coloana face parte dintr-o cheie PRIMARY KEY
UNIQUE_KEY_FLAG	Coloana face parte dintr-un index UNIQUE
UNSIGNED_FLAG	Coloana are atributul UNSIGNED
ZEROFILL_FLAG	Coloana are atributul ZEROFILL

Câteva constante de tip `flags` indică tipuri de coloane, nu atribute de coloane; acestea nu mai sunt utilizate în prezent, deoarece trebuie să folosiți `field-type` pentru a determina tipul coloanei. Tabelul F.3 prezintă aceste constante ieșite din uz.

Tabelul F.3 Valori ieșite din uz ale membrului `flags` al structurii `MYSQL_FIELD`

Valoarea membrului `flags`	Semnificație
BLOB_FLAG	Coloana conține valori BLOB
ENUM_FLAG	Coloana conține valori ENUM
SET_FLAG	Coloana conține valori SET
TIMESTAMP_FLAG	Coloana conține valori TIMESTAMP

- `decimals`

  Numărul de zecimale pentru coloanele numerice; `decimals` este zero pentru coloanele care nu conțin valori numerice. De exemplu, o coloană `DECIMAL(8,3)` are o valoare `decimals` egală cu 3, în timp ce aceeași valoare este egală cu 0 în cazul coloanei `BLOB`.

Macrocomenzi accesorii

`mysql.h` conține câteva macrocomenzi care vă permit să testați membrii structurii `MYSQL_FIELDS` într-un mod mai convenabil. Funcția `IS_NUM()` testează membrul `type`; celelalte funcții prezentate aici testează membrul `flags`.

`IS_NUM()` este adevărat (diferit de zero) dacă valorile din coloană sunt de tip numeric:

```
if (IS_NUM (field->type))
    printf("Campul %s este numeric\n", field->name);
```

`IS_PRI_KEY()` are valoarea „adevărat" dacă o coloană face parte dintr-o cheie primară:

```
if (IS_PRI_KEY (field->flags))
    printf("Campul %s face parte dintr-o cheie primara\n", field->name);
```

`IS_NOT_NULL()` are valoarea „adevărat" dacă o coloană nu poate conține valori `NULL`:

```
if (IS_NOT_NULL (field->flags))
    printf("Valorile din campul %s nu pot fi NULL\n", field->name);
```

Funcția `IS_BLOB()` are valoarea „adevărat" în cazul în care coloana este de tip `BLOB` sau `TEXT`. Cu toate acestea, macrocomanda testează bitul membrului `flags` `BLOB_FLAG` scos din uz, deci și `IS_BLOB()` este scoasă din uz.

Funcții ale interfeței API C

Funcțiile din biblioteca client pot fi clasificate în următoarele categorii:

- Rutine de gestionare a conexiunii, pentru stabilirea și terminarea conexiunilor cu serverul
- Rutine de stare și de raportare a erorilor pentru obținerea de mesaje și coduri de eroare
- Rutine de construcție și execuție a interogărilor pentru construirea interogărilor și trimiterea acestora la server
- Rutine de prelucrare a seturilor de rezultate, pentru tratarea rezultatelor interogărilor care returnează date
- Rutine de informare, care furnizează informații despre client, server, versiunea protocolului și conexiunea curentă
- Rutine de administrare pentru controlul modului de operare a serverului
- Rutine de depanare, pentru generarea informațiilor de depanare
- Rutine scoase din uz, care acum sunt considerate depășite

Funcțiile din biblioteca client aferente fiecăreia dintre aceste categorii sunt descrise detaliat în secțiunile următoare, în ordine alfabetică, în cadrul fiecărei categorii. Anumiți parametri apar de mai multe ori în descrierile funcțiilor și au o semnificație standard:

- `conn` este un pointer spre variabila `MYSQL` de tratare a conexiunilor pentru o conexiune cu serverul.
- `res_set` este un pointer spre o structură cu seturi de rezultate `MYSQL_RES`.
- `field` este un pointer spre o structură `MYSQL_FIELD` cu informații privind coloanele.
- `row` este un rând de date `MYSQL_ROW` dintr-un set de rezultate.

Din motive de concizie, acolo unde acești parametri nu sunt menționați în descrierea funcțiilor în care apar, puteți lua în considerare descrierile menționate anterior.

Dacă nu este specificat altceva, puteți presupune că orice funcție dată există în biblioteca client începând de la versiunea MySQL 3.21.0.

Rutine de gestionare a conexiunii

Aceste funcții vă permit să stabiliți și să terminați conexiuni cu un server, să stabiliți opțiuni care influențează modul de stabilire a conexiunii, precum și să restabiliți conexiuni expirate.

- `my_bool`

  `mysql_change_user` `(MYSQL *conn, char *user_name, char*password, char *db_name);`

  Modifică utilizatorul și baza de date prestabilită ale conexiunii specificate de `conn`. Baza de date devine prestabilită pentru referințele la tabele care nu includ un specificator pentru baza de date. Dacă `db_name` este `NULL`, nu este selectată nici o bază de date prestabilită.

`mysql_change_user()` returnează „adevărat" dacă utilizatorul are permisiunea de a se conecta la server și, dacă a fost specificată o bază de date, de a obține accesul la baza de date. În caz contrar, funcția eșuează și utilizatorul și baza de date curente rămân nemodificate.

Funcția `mysql_change_user()` a fost introdusă în versiunea MySQL 3.23.3.

- `void`

  `mysql_close``(MYWQL *conn);`

  Închide conexiunea specificată de `conn`. Apelați această rutină când ați terminat o sesiune server. În cazul în care variabila de tratare a conexiunii a fost alocată automat de către `mysql_init()`, funcția `mysql_close()` anulează alocarea.

  Nu apelați funcția `mysql_close()` dacă încercarea de a deschide o conexiune a eșuat.

- `MYSQL *`

  `mysql_init``(MYSQL *conn);`

  Inițializează o variabilă de tratare a conexiunilor și returnează un pointer la aceasta. În cazul în care `conn` indică spre o structură `MYSQL` existentă, `mysql_init()` inițializează acea variabilă și returnează adresa acesteia:

```
MYSQL conn_struct, *conn;
conn = mysql_init (&conn_struct);
```

  În cazul în care `conn` este `NULL`, `mysql_init()` alocă o nouă variabilă de tratare, o inițializează și îi returnează adresa:

```
MYSQL *conn;
conn = mysql_init (NULL);
```

  Dacă funcția `mysql_init()` eșuează, returnează `NULL`. Aceasta se poate întâmpla dacă `mysql_init()` nu poate aloca o nouă variabilă de tratare.

  Dacă `mysql_init()` alocă variabila de tratare, `mysql_close()` anulează automat alocarea acesteia atunci când închideți conexiunea.

  Funcția `mysql_init()` a fost introdusă în MySQL versiunea 3.22.1.

- `int`

  `mysql_options``(MYSQL *conn, enum mysql_option option, char *arg);`

  Această funcție vă permite să ajustați comportarea conexiunii cu mai multă precizie decât folosind numai funcția `mysql_real_connect()`. Apelați această funcție după `mysql_init()` și înainte de `mysql_real_connect()`. Puteți apela funcția `mysql_options()` de mai multe ori, dacă doriți să configurați mai multe opțiuni.

  Argumentul `option` specifică opțiunea de conexiune pe care doriți să o configurați. Informațiile suplimentare necesare pentru a configura opțiunea, dacă există, sunt specificate de argumentul `arg`. (Rețineți că `arg` este un pointer.) `arg` este `NULL` dacă nu sunt necesare informații suplimentare.

  Sunt disponibile următoarele opțiuni:

- `MYSQL_INIT_COMMAND`

  Specifică o interogare care se va executa după conectarea la server. `arg` este un șir terminat cu nul care conține interogarea. Interogarea este executată și după reconectare (de exemplu, dacă apelați funcția `mysql_ping()`). Orice set de rezultate returnat de interogare este ignorat.

- `MYSQL_OPT_COMPRESS`

  Specifică faptul că respectiva conexiune trebuie să utilizeze protocolul client/server comprimat, dacă serverul îl acceptă. `arg` este `NULL`.

  De asemenea, se poate specifica utilizarea comprimării atunci când apelați funcția `mysql_real_connect()`.

- `MYSQL_OPT_CONNECT_TIMEOUT`

  Indică intervalul de expirare a conexiunii, exprimat în secunde. `arg` este un pointer spre un întreg fără semn care conține valoarea de expirare.

- `MYSQL_OPT_NAMED_PIPE`

  Indică faptul că acea conexiune la server trebuie să folosească un canal denumit. `arg` este `NULL`. Această opțiune este destinată numai clienților Windows 95/98/NT și numai pentru conexiunile la serverele Windows NT.

- `MYSQL_READ_DEFAULT_FILE`

  Specifică un fișier cu opțiuni care va fi citit pentru determinarea parametrilor de conexiune. Opțiunile sunt citite din grupul `[client]` din fișier.

- `MYSQL_READ_DEFAULT_GROUP`

  Specifică un grup care se va citi din fișierul cu opțiuni denumit cu opțiunea `MYSQL_READ_DEFAULT_FILE`. Grupul denumit este citit alături de grupul `[client]`. Dacă nu este denumit nici un fișier cu opțiuni, biblioteca client caută fișierele cu opțiuni standard și le citește.

  Dacă apelați `mysql_options()` de mai multe ori pentru a configura o opțiune dată, `mysql_real_connect()` folosește valoarea specificată cel mai recent pentru opțiunea respectivă.

  Funcția `mysql_options()` returnează zero în caz de reușită, respectiv o valoare diferită de zero dacă valoarea `option` este necunoscută.

  Într-un fișier cu opțiuni se pot folosi următoarele opțiuni (în grupul `[client]` sau într-un grup pe care-l specificați cu opțiunea `MYSQL_READ_DEFAULT_GROUP`):

```
compress
database=nume_db
debug
host=nume_gazda
init-command=interogare
password=parola_dv
pipe
port=numar_port
return-found-rows
socket=nume_soclu
timeout=secunde
user=nume_utilizator
```

  Apelurile la funcția `mysql_options()` din exemplul următor configurează opțiunile de conexiune astfel încât `mysql_real_connect()` se conectează prin intermediul unui canal denumit, folosind protocolul comprimat și un interval de expirare de 10 secunde. Pentru a obține informații de la grupurile `[client]` și `[mygroup]`, funcția citește fișierul `C:\my.cnf.special`. La stabilirea conexiunii, se va executa o instrucțiune `SET SQL_BIG_TABLES`.

```
MYSQL *conn;
unsigned int expirare;

if ((conn = mysql_init (NULL)) == NULL)
        ... tratarea erorii ...
expirare = 10;
mysql_options (conn, MYSQL_OPT_CONNECT_TIMEOUT, (char *) &timeout);
mysql_options (conn, MYSQL_OPT_COMPRESS, NULL);
mysql_options (conn, MYSQL_NAMED_PIPE, NULL);
mysql_options (conn, MYSQL_READ_DEFAULT_FILE, "C:\my.cnf.special");
mysql_options (conn, MYSQL_READ_DEFAULT_GROUP, "grupmeu");
mysql_options (conn, MYSQL_INIT_COMMAND, "SET SQL_BIG_TABLES=1");
if (mysql_real_connect (conn, ...) == NULL)
        ... tratarea erorii ...
```

  Funcția `mysql_options()` a fost introdusă în MySQL 3.22.1. `MYSQL_INIT_COMMAND`, `MYSQL_READ_DEFAULT_FILE` și `MYSQL_READ_DEFAULT_GROUP` au fost introduse în versiunea MySQL 3.22.10.

- `int`

  `mysql_ping``(MYSQL *conn);`

  Verifică dacă conexiunea indicată de `conn` este încă funcțională. Dacă nu este, funcția `mysql_ping()` se conectează din nou, folosind aceiași parametri care au fost utilizați inițial pentru stabilirea conexiunii. Astfel, nu trebuie să apelați funcția `mysql_ping()` fără a fi apelat cu succes, în prealabil, funcția `mysql_real_connect()`.

  Funcția `mysql_ping()` a fost introdusă în versiunea MySQL 3.22.1.

- `MYSQL *`

```
mysql_real_connect (MYSQL *conn, char *nume_gazda,
                    char *nume_utilizator, char *parola,
                    char *nume_db, unsigned int numar_port,
                    char *nume_soclu, unsigned int flags);
```

  Se conectează la un server și returnează un pointer spre variabila de tratare a conexiunii. `conn` trebuie să fie un pointer spre o variabilă existentă de tratare a conexiunii care a fost inițializată de `mysql_init()`. Adresa variabilei de tratare este valoarea returnată în caz de conexiune reușită. Dacă se produce o eroare, funcția returnează `NULL`.

  În cazul în care tentativa de conexiune eșuează, puteți transmite valoarea variabilei de tratare `conn` funcțiilor `mysql_errno()` și `mysql_error()` pentru a obține informații despre eroare. Totuși, nu trebuie să transferați valoarea `conn` nici unei alte rutine din biblioteca client care presupune că variabila de tratare reflectă o conexiune reușită.

  `nume_gazda` este serverul la care se va stabili conexiunea. Tabelul F.4 prezintă modul în care clientul încearcă să se conecteze pentru diferite valori `nume_gazda` pentru clienți

UNIX și Windows. (Dacă vă conectați folosind un soclu UNIX sau un canal denumit, parametrul `nume_soclu` specifică numele soclului sau al canalului.)

Tabelul F.4 Tipuri de conexiune client, în funcție de tipul numelui gazdei serverului

Numele gazdei	Tip de conexiune UNIX	Tip de conexiune Windows
`nume_gazda`	Conexiune TCP/IP cu gazda denumită	Conexiune TCP/IP cu gazda denumită
Număr IP	Conexiune TCP/IP cu gazda denumită	Conexiune TCP/IP cu gazda denumită
`localhost`	Conexiune prin soclu UNIX cu gazda locală	Conexiune TCP/IP cu gazda denumită.
.	Nu este valabil	Conexiune prin canal denumit cu gazda locală
`NULL`	Conexiune prin soclu UNIX la gazda locală	Conexiune TCP/IP la gazda locală, cu excepția că în Windows NT se încearcă mai întâi o conexiune prin canal denumit înainte de a se recurge la TCP/IP

`nume_utilizator` este numele dumneavoastră de utilizator MySQL. Dacă acesta este `NULL`, biblioteca client trimite un nume prestabilit. Sub UNIX, numele prestabilit este numele dumneavoastră de deschidere a sesiunii de lucru. Sub Windows, numele prestabilit este numele dumneavoastră așa cum este specificat în variabila de mediu `USER`, dacă este activată, respectiv `"ODBC"` în caz contrar.

`parola` este parola dumneavoastră. Dacă este `NULL`, vă veți putea conecta numai dacă tabelul de acordare `user` conține o intrare care corespunde numelui dumneavoastră de utilizator și gazdei de la care vă conectați și dacă intrarea respectivă conține o parolă vidă.

`nume_db` este baza de date care se va utiliza. Dacă este `NULL`, nu va fi selectată nici o bază de date inițială.

`numar_port` este numărul portului folosit pentru conexiunile TCP/IP. Dacă este `0`, se va folosi numărul de port prestabilit.

Numărul portului și numele soclului sunt folosite în conformitate cu valoarea parametrului `nume_gazda`. Dacă vă conectați la gazda locală, funcția `mysql_real_connect()` încearcă o conexiune folosind un soclu de domeniu UNIX (sub UNIX), respectiv un canal denumit (sub Windows). Altfel, se conectează folosind TCP/IP.

`flags` poate fi `0` pentru a nu specifica nici o opțiune, respectiv poate lua una dintre valorile prezentate în tabelul F.5. Opțiunile influențează modul de operare a serverului.

Tabelul F.5 Valorile `flags` pentru funcția `mysql_real_connect()`

Valoarea indicatorului	Efectul asupra modului de operare a serverului
`CLIENT_FOUND_ROWS`	Pentru interogările `UPDATE`, returnează numărul de rânduri care au corespuns criteriilor de căutare, nu numărul rândurilor modificate.
`CLIENT_NO_SCHEMA`	Nu permite sintaxa *`nume_db.nume_tabel.nume_coloana`*
`CLIENT_COMPRESS`	Folosește protocolul de comunicații comprimat, dacă serverul îl acceptă.
`CLIENT_ODBC`	Tratează clientul ca pe un client ODBC.

Dacă specificați `CLIENT_NO_SCHEMA`, serverul permite utilizarea în interogări numai a referințelor de tipul *`nume_tabel.nume_coloana`*, *`nume_tabel`* sau *`nume_coloana`*.

Valorile indicatorilor reprezintă valori de biți, deci le puteți combina folosind caracterele | sau +; de exemplu `CLIENT_COMPRESS|CLIENT_ODBC` sau `CLIENT_COMPRESS+CLIENT_ODBC`.

Funcția `mysql_real_connect()` a fost introdusă în versiunea MySQL 3.21.10. Parametrul `nume_db` a fost adăugat în versiunea 3.22.0. Utilizarea funcției `mysql_init()` pentru inițializarea argumentului `MYSQL` a început de la versiunea MySQL 3.22.1.

Funcții de stare și de raportare a erorilor

Funcțiile din această secțiune vă permit să determinați și să raportați cauzele erorilor:

- `unsigned int`

  `mysql_errno``(MYSQL *conn);`

  Returnează un cod de eroare pentru rutina din biblioteca client cel mai recent invocată care a returnat o stare. Codul de eroare este `0` dacă nu s-a produs nici o eroare, respectiv un cod diferit de zero în caz contrar. Codurile de eroare posibile sunt enumerate în fișierele antet MySQL `errmsg.h` și `mysqld.h`.

```
if (mysql_errno(conn) = = 0)
    printf("Totul este in regula\n");
else
    printf("S-a intamplat ceva!\n");
```

  Funcția `mysql_errno()` a fost introdusă în versiunea MySQL 3.21.7.

- `char *`

  `mysql_error``(MYSQL *conn);`

  Returnează un șir terminat cu nul care conține un mesaj de eroare pentru rutina din biblioteca client cel mai recent invocată care a returnat o stare. Valoarea returnată este șirul vid dacă nu s-a produs nici o eroare (este șirul de lungime zero `""`, nu un pointer `NULL`). Deși în mod normal apelați funcția `mysql_error()` după ce ați aflat deja că s-a produs o eroare, se poate folosi chiar valoarea returnată pentru a detecta apariția unei erori:

```
if (mysql_errno(conn)[0] = = '\0') /* sirul este vid? */
    printf("Totul este in regula\n");
else
    printf("S-a intamplat ceva!\n");
```

  Funcția `mysql_error()` a fost introdusă în versiunea MySQL 3.21.7 (poate ceva mai târziu).

Rutine de construcție și execuție a interogărilor

Funcțiile din această secțiune vă permit să trimiteți interogări serverului.

`mysql_escape_string()` vă ajută să construiți interogări prin modificarea semnificației anumitor caractere care necesită un tratament special.

- `unsigned int`
  `mysql_escape_string``(char *to_str, char *from_str,` unsigned int from_len);

  Codifică un șir care poate conține caractere speciale, astfel încât să poată fi folosit într-o instrucțiune SQL. Tabelul F.6 enumeră caracterele care sunt considerate speciale și codificările acestora.

Tabelul F.6 Codificări de caractere pentru funcția `mysql_escape_string()`

Caracter special	Codificare
NUL (ASCII 0)	\0 (backslash-0, nu backslash-nul)
Backslash	\\ (backslash-backslash)
Ghilimele simple	\' (backslash-ghilimele simple)
Ghilimele duble	\" (backslash-ghilimele duble)
Linie nouă	\n (backslash-n)
Retur de car	\r (backslash-r)
Ctrl-Z	\Z (backslash-Z)

Bufferul care va fi codificat este specificat sub formă de șir numărat. `from_str` indică spre bufferul respectiv, iar `from_len` indică numărul de octeți din acesta.

Funcția `mysql_escape_string()` scrie rezultatul în bufferul indicat de `to_str` și adaugă un octet nul. `to_str` trebuie să indice spre un buffer existent, care să aibă minimum `(from_len*2)+1` octeți lungime. (În cel mai rău caz, fiecare caracter din `from_str` trebuie codificat sub forma unei secvențe de două caractere; de asemenea, mai aveți nevoie de spațiu pentru nulul de terminare.)

Funcția `mysql_escape_string()` returnează lungimea șirului codificat, fără a număra octetul nul de terminare.

Șirul codificat rezultant nu conține caractere nul interne, dar se termină cu nul, deci îl puteți folosi cu funcții precum `strlen()` sau `strcat()`.

Când scrieți șiruri literale în programul dumneavoastră, fiți atent să nu confundați convențiile lexicale de modificare a semnificației caracterelor specifice limbajului C cu codificările executate de funcția `mysql_escape_string()`. Să examinăm următorul exemplu și rezultatul determinat de acesta:

```
to_len = mysql_escape_string (to_str, "\0\\\'\"\n\r\032", 7);
printf("to_len = %d, to_str = %s/n", to_len, to_str);
```

Datele de ieșire sunt următoarele:

```
to_len = 14, to_str = \0\\\'\"\n\r\Z
```


Valoarea afișată a șirului `to_str` seamănă foarte mult cu șirul specificat ca al doilea argument al apelului la funcția `mysql_escape_string()`, dar, în realitate, este foarte diferit de acesta.

- `int`
  `mysql_query``(MYSQL *conn, char *sir_interogare);`

  Dacă este dată o interogare cu un șir care se termină cu nul, `mysql_query()` trimite interogarea serverului, pentru a fi executată. Șirul nu trebuie să conțină date binare; în particular, nu trebuie să conțină octeți nuli, pe care funcția `mysql_query()` îi interpretează ca fiind sfârșitul interogării. Dacă interogarea dumneavoastră conține date binare, folosiți în schimb funcția `mysql_real_query()`.

  Interogarea trebuie să fie alcătuită dintr-o singură instrucțiune SQL și nu trebuie să se încheie cu punct și virgulă (`;`) sau cu `\g`; acestea reprezintă convenții ale programului client `mysql`, nu ale bibliotecii client.

  Funcția `mysql_query()` returnează zero în caz de reușită și o valoare diferită de zero în caz de eșec. O interogare reușită este una pe care serverul o acceptă ca fiind corectă și pe care o execută fără eroare. În caz de succes, nu se menționează nimic despre numărul rândurilor afectate sau returnate.

- `int`
  `mysql_real_query``(MYSQL *conn, char *sir_interogare, unsigned int lungime);`

  Dacă este dată o interogare specificată sub formă de șir numărat, funcția `mysql_real_query()` trimite interogarea la server, pentru a fi executată. Șirul poate conține date binare (inclusiv octeți nuli). Textul interogării este dat de `sir_interogare`, iar lungimea este indicată de `lungime`.

  Interogarea trebuie să fie alcătuită dintr-o singură instrucțiune SQL și nu trebuie să se încheie cu punct și virgulă (`;`) sau cu `\g`; acestea reprezintă convenții ale programului client `mysql`, nu ale bibliotecii client.

  Funcția `mysql_real_query()` returnează zero în caz de reușită și o valoare diferită de zero în caz de eșec. O interogare reușită este una pe care serverul o acceptă ca fiind corectă și care se execută fără eroare. În caz de succes, nu se menționează nimic despre numărul rândurilor afectate sau returnate.

- `int`
  `mysql_select_db``(MYSQL *conn, char *nume_db);`

  Selectează baza de date denumită de `nume_db` drept bază de date curentă, care devine bază de date prestabilită pentru referințele la tabele care nu conțin nici un specificator explicit al bazei de date. Trebuie să aveți permisiunea de a obține accesul la baza de date; în caz contrar, funcția `mysql_select_db()` eșuează.

  Funcția `mysql_select_db()` returnează zero în caz de succes, respectiv o valoare diferită de zero în caz de eșec.

Rutine de prelucrare a setului de rezultate

Când o interogare produce un set de rezultate, funcțiile din această secțiune vă permit să regăsiți setul și să obțineți acces la conținutul său. Funcțiile `mysql_store_result()` și `mysql_use_result()` creează setul de rezultate, iar una din ele trebuie apelată înainte de utilizarea oricăror alte funcții din această secțiune. Tabelul F.7 compară cele două funcții.

Tabelul F.7 O comparație între funcțiile `mysql_store_result()` și `mysql_use_result()`.

`mysql_store_result()`	**`mysql_use_result()`**
Toate rândurile din setul de rezultate sunt preluate chiar de funcția `mysql_store_result()`.	`mysql_use_result()` inițializează setul de rezultate, dar lasă regăsirea rândurilor în seama funcției `mysql_fetch_row()`
Folosește o cantitate mai mare de memorie; toate rândurile sunt stocate la client.	Folosește o cantitate mai redusă de memorie; este stocat fiecare rând în parte.
Este mai lentă, datorită suprasarcinii implicate de alocarea de memorie pentru întregul set de rezultate.	Este mai rapidă, deoarece este necesară alocarea de memorie numai pentru rândul curent.
Valoarea `NULL` returnată de funcția `mysql_fetch_row()` indică sfârșitul setului de rezultate; nu indică niciodată o eroare.	Valoarea `NULL` returnată de funcția `mysql_fetch_row()` indică sfârșitul setului de rezultate sau o eroare, deoarece o pană de comunicare poate afecta regăsirea înregistrării curente.
Funcția `mysql_num_rows()` poate fi apelată în orice moment după apelarea funcției `mysql_store_result()`.	Funcția `mysql_num_rows()` returnează un număr corect de rânduri numai după preluarea tuturor rândurilor.
Funcția `mysql_affected_rows()` este sinonimă cu `mysql_num_rows()`.	Funcția `mysql_affected_rows()` nu poate fi folosită.
Accesul aleator la rândurile din setul de rezultate este posibil cu funcțiile `mysql_data_seek()`, `mysql_row_seek()` și `mysql_row_tell()`.	Nu este permis accesul aleator la setul de rezultate; rândurile trebuie prelucrate în ordinea în care sunt returnate de server; funcțiile `mysql_data_seek()`, `mysql_row_seek()` și `mysql_row_tell()` nu trebuie folosite.
Nu trebuie să apelați deloc funcția `mysql_fetch_row()`, deoarece toate rândurile au fost deja regăsite de client.	*Trebuie* să apelați funcția `mysql_fetch_row()` pentru a prelua toate rândurile; în caz contrar, rândurile „rămase" se vor infiltra în rezultatul următoarei interogări, determinând o eroare de sincronizare.
Tabelele sunt blocate la citire pentru un interval de timp strict necesar preluării rândurilor de date.	Tabelele pot rămâne în starea „blocat pentru citire" dacă clientul se oprește în mijlocul unei operații de regăsire, blocând orice client care încearcă să le modifice.
Membrul `max_length` al structurilor de seturi de rezultate `MYSQL_FIELD` primește cea mai lungă valoare existentă în setul de rezultate pentru coloanele din set.	`max_length` nu este configurat la nici o valoare semnificativă, deoarece aceasta nu poate fi cunoscută decât după regăsirea tuturor rândurilor.

- `my_ulonglong`

  `mysql_affected_rows``(MYSQL *conn);`

  Returnează numărul de rânduri modificate de cea mai recentă interogare `DELETE`, `INSERT`, `REPLACE` sau `UPDATE`. Pentru asemenea interogări, funcția `mysql_affected_rows()` poate fi apelată imediat după un apel reușit la funcția `mysql_query()`. De asemenea, puteți apela această funcție după ce ați emis o instrucțiune care returnează rânduri. În acest caz, funcția se comportă în același mod ca și `mysql_num_rows()` și este supusă acelorași restricții pentru care valoarea are sens, precum și la restricția suplimentară conform căreia funcția `mysql_affected_rows()` nu este niciodată semnificativă atunci când folosiți `mysql_use_result()`.

  Funcția `mysql_affected_rows()` returnează zero dacă nu a fost emisă nici o interogare sau dacă interogarea returnează rânduri, dar nu a fost selectat nici un rând. O valoare returnată mai mare decât zero indică numărul rândurilor modificate (pentru `DELETE`, `INSERT`, `REPLACE` și `UPDATE`) respectiv returnate (pentru interogările care returnează rânduri). O valoare returnată egală cu –1 indică fie o eroare, fie că ați apelat (în mod greșit) funcția `mysql_affected_rows()` după ce ați emis o interogare care returnează rânduri, dar înainte de a regăsi efectiv setul de rezultate. Deoarece funcția `mysql_affected_rows()` returnează o valoare fără semn, trebuie să efectuați comparația prin conversia rezultatului său într-o valoare cu semn:

  ```
  if ((long) mysql_affected_rows(conn) = = -1)
          fprintf(stderr, "Eroare!\n");
  ```

  Dacă ați specificat că clientul trebuie să returneze numărul de rânduri care au corespuns criteriilor de căutare în cazul interogărilor `UPDATE`, funcția `mysql_affected_rows()` returnează valoarea respectivă, nu numărul rândurilor efectiv modificate. (MySQL nu actualizează un rând în cazul în care coloanele care urmează a fi modificate sunt identice cu noile valori.) Această comportare poate fi selectată fie prin specificarea opțiunii `return-found-rows` într-un fișier cu opțiuni, fie prin transferarea opțiunii `CLIENT_FOUND_ROWS` în parametrul `flags` către funcția `mysql_real_connect()`.

  Vezi nota despre afișarea valorilor de tip `my_ulonglong` din secțiunea „Tipuri de date scalare".

- `void`

  `mysql_data_seek``(MYSQL_RES *res_set, unsigned int offset);`

  Caută într-un anumit rând din setul de rezultate. Valoarea parametrului `offset` poate varia între 0 și `mysql_num_rows(res_set)-1`. Rezultatele nu pot fi anticipate dacă `offset` nu se găsește în acest domeniu.

  `mysql_data_seek()` impune ca întregul set de rezultate să fie regăsit, deci puteți folosi această funcție numai dacă setul de rezultate a fost creat cu funcția `mysql_store_result()`, nu cu funcția `mysql_use_result()`.

- `MYSQL_FIELD *`

  `mysql_fetch_field``(MYSQL_RES *res_set);`

  Returnează o structură care conține informații (metadate) despre o coloană din setul de rezultate. După ce ați executat cu succes o interogare care returnează rânduri,

primul apel la funcția `mysql_fetch_field()` returnează informații despre prima coloană. Apelurile ulterioare returnează informații despre coloanele următoare, respectiv `NULL` când nu a mai rămas nici o coloană.

De asemenea, puteți apela funcția `mysql_field_tell()` pentru a determina poziția coloanei curente, respectiv funcția `mysql_field_seek()` pentru a selecta o anumită coloană care va fi returnată de următorul apel la funcția `mysql_fetch_field()`.

Exemplul următor caută prima structură `MYSQL_FIELD`, apoi preia următoarele structuri de informații privind coloanele:

```
MYSQL_FIELD         *field;
unsigned int        i;

mysql_field_seek(res_set, 0);
for (i = 0; i < mysql_num_fields(res_set); i++)
{
        field = mysql_fetch_field(res_set);
        printf("column %u: name = %s max_length = %lu\n",
                i, field->name, field->max_length);
}
```

- `MYSQL_FIELD *`

`mysql_fetch_fields``(MYSQL_RES res_set);`

Returnează un tablou cu toate structurile care conțin informații despre coloane aferente setului de rezultate. La acestea se poate obține accesul astfel:

```
MYSQL_FIELD         *field;
unsigned int        i;

field = mysql_fetch_fields(res_set);
for (i = 0; i < mysql_num_fields(res_set); i++)
{
        printf("column %u: name = %s max_length = %lu\n",
                i, field[i].name, field[i].max_length);
}
```

Comparați acest listing cu exemplul prezentat pentru funcția `mysql_fetch_field()`. Rețineți că, deși ambele funcții returnează valori de același tip, aceste valori sunt accesibile folosind sintaxe ușor diferite pentru fiecare funcție în parte.

- `MYSQL_FIELD *`

`mysql_fetch_field_direct``(MYSQL_RES *res_set, unsigned int field_num);`

Dacă este dat indexul unei coloane, returnează structura de informații aferentă acelei coloane. Valoarea parametrului `field_num` poate varia între `0` și `mysql_num_fields()-1`. Rezultatele sunt imprevizibile dacă `field_num` nu se încadrează în domeniu.

În exemplul următor, se obține accesul direct la structurile `MYSQL_FIELD`:

```
MYSQL_FIELD         *field;
unsigned int        i;
```



```
for (i = 0; i < mysql_num_fields(res_set); i++)
{
        field = mysql_fetch_field_direct(res_set, i);
        printf("column %u: name = %s max_length = %lu\n",
                i, field->name, field->max_length);
}
```

Funcția `mysql_fetch_field_direct()` nu funcționează în mod corespunzător anterio variantei MySQL 3.23.

- `unsigned long *`

`mysql_fetch_lengths``(MYSQL_RES *res_set);`

Returnează un pointer la un tablou de valori `unsigned long` care reprezintă lungimil coloanelor din rândul curent al setului de rezultate. Trebuie să apelați funcți `mysql_fetch_lengths()` de fiecare dată când apelați `mysql_fetch_row()`; în caz con trar, lungimile vor fi nesincronizate cu valorile datelor.

Lungimea valorilor `NULL` este zero, dar o lungime zero nu indică prin prezența sa o valoar `NULL`. Deoarece un șir vid are de asemenea o lungime zero, trebuie să verificați dacă valoarea de date este un pointer `NULL`, pentru a face diferența între cele două cazuri.

Exemplul următor afișează lungimile și valorile pentru rândul curent, afișând `NUL` dacă valoarea este `NULL`:

```
unsigned long *length;

length = mysql_fetch_lengths(res_set);
for(i = 0; i < mysql_num_fields(res_set); i++)
{
        printf("lungimea este %lu, valoarea este %s\n",
                length[i], (row[i] != NULL ? row[i] : "NULL"));
}
```

Funcția `mysql_fetch_lengths()` a apărut în versiunea MySQL 3.20.5. Anterior versiu nii MySQL 3.22.7, tipul valorii returnate de funcția `mysql_fetch_lengths()` er `unsigned int`.

- `MYSQL_ROW`

`mysql_fetch_row``(MYSQL_RES *res_set);`

Returnează un pointer la următorul rând al setului de rezultate, reprezentat sub form unui tablou de șiruri (cu excepția faptului că valorile `NULL` din coloane sunt reprezen tate sub formă de pointeri `NULL`). A `i`-a valoare din rând este al `i`-lea membru a tabloului de valori.

Valorile pentru toate tipurile de date, chiar și pentru tipurile numerice, sunt returnat sub formă de șiruri. Dacă doriți să efectuați un calcul numeric cu o valoare, trebuie s convertiți personal valoarea, folosind, de exemplu, funcțiile `atoi()` sau `atof()`.

Funcția `mysql_fetch_row()` returnează `NULL` atunci când nu mai există rânduri în setu de rezultate. (Dacă folosiți `mysql_use_result()` pentru a iniția o regăsire rând cu rân a setului de rezultate, `mysql_fetch_row()` va returna de asemenea `NULL` dacă se pro duce o eroare de comunicații.)

Valorile de date se termină cu nul, dar nu tratați valorile care pot conține date binare drept șiruri terminate cu nul. Acestea trebuie tratate ca șiruri numerotate. (Pentru a determina lungimile valorilor din coloane, apelați funcția `mysql_fetch_lengths()`.)

Programul următor prezintă modul de parcurgere ciclică a unui rând cu valori de date și de a determina dacă fiecare valoare este sau nu `NULL`:

```
MYSQL_ROW          row;
unsigned int       i;

while ((row = mysql_fetch_row(res_set)) != NULL)
{
        for (i = 0; i < mysql_num_fields(res_set); i++)
        {
                printf("coloana %u: valoarea este %s\n",
                        i, (row[i] = = NULL ? "NULL" : "not NULL"));
        }
}
```

Pentru a determina tipurile valorilor din coloane, folosiți metadatele despre coloane stocate în structurile `MYSQL_FIELD` de informații privind coloanele, obținute prin apelarea uneia din funcțiile `mysql_fetch_field()`, `mysql_fetch_fields()` sau `mysql_fetch_field_direct()`.

- `unsigned int`

  `mysql_field_count``(MYSQL *conn);`

Returnează numărul de coloane pentru cea mai recentă interogare asupra conexiunii date. Această funcție este folosită de obicei când funcția `mysql_store_result()` sau `mysql_use_result()` returnează `NULL`. Funcția `mysql_field_count()` arată dacă ar fi fost necesară sau nu returnarea unui set de rezultate. O valoare returnată egală cu 0 indică absența unui set de rezultate și absența erorilor. O valoare diferită de zero arată că funcția aștepta coloane și că, deoarece nu a fost returnată nici o coloană, s-a produs o eroare.

Exemplul următor ilustrează modul de utilizare a funcției `mysql_field_count()` pentru detecția erorilor:

```
res_set = mysql_store_result (conn);
if (res_set == NULL)  /* nu a fost returnat nici un set de rezultate */
{
        /*
         * lipsa unui set de rezultate inseamna ca s-a produs o
         * eroare sau ca nu a fost returnat nici un set de rezultate?
         */
        if (mysql_field_count (conn) > 0)
        {
                /*
                 * se astepta un set de rezultate, dar
                 * mysql_store_result() nu a returnat nimic;
                 * aceasta inseamna ca s-a produs o eroare
                 */
                print_error (conn,
                  "Problema la prelucrarea setului de rezultate");
        }
        else
        {
                /*
                 * nu a fost returnat nici un set de
                 * rezultate; interogarea nu a returnat date
                 * (nu a fost de tip SELECT, SHOW, DESCRIBE
                 * sau EXPLAIN), deci raporteaza numarul de
                 * randuri afectate de interogare
                 */
                printf ("%lu randuri afectate\n",
                  (unsigned long) mysql_affected_rows (conn));
        }
}
else /* a fost returnat un set de rezultate */
{
        /* prelucreaza randurile aici */
        mysql_free_result (res_set);
}
```


Funcția `mysql_field_count()` a fost introdusă în versiunea MySQL 3.22.24. Anterior acestei versiuni, funcția `mysql_nume_fields()` era folosită în același scop. Pentru a scrie programe care să funcționeze cu orice versiune a sistemului MySQL, includeți următorul fragment în orice fișier care folosește funcția `mysql_field_count()`:

```
#if !defined(MYSQL_VERSION_ID) || MYSQL_VERSION_ID<32224
#define mysql_field_count mysql_num_fields
#endif
```

Această secvență de instrucțiuni mapează funcția `mysql_field_count()` pe funcția `mysql_num_fields()`atunci când codul sursă este compilat sub versiuni mai vechi ale programului MySQL.

- `MYSQL_FIELD_OFFSET`

  `mysql_field_seek``(MYSQL_RES *res_set, MYSQL_FIELD_OFFSET offset);`

Caută structura cu informații despre coloane specificată de `offset`. Următorul apel la funcția `mysql_fetch_field()` va returna structura cu informații pentru coloana dată. `offset` *nu* este un index de coloană; este o valoare `MYSQL_FIELD_OFFSET` obținută printr-un apel anterior la una din funcțiile `mysql_field_tell()` sau `mysql_field_seek()`.

Pentru a reinițializa la prima coloană, transferați o valoare `offset` egală cu 0.

- `MYSQL_FIELD_OFFSET`

  `mysql_field_tell``(MYSQL_RES *res_set);`

Returnează offset-ul structurii curente cu informații privind coloanele. Această valoare poate fi transferată funcției `mysql_field_seek()`.

- `void`
  `mysql_free_result``(MYSQL_RES *res_set);`

  Anulează alocarea memoriei folosite pentru setul de rezultate. Trebuie să apelaţi funcţia `mysql_free_result()` pentru fiecare set de rezultate pe care îl utilizaţi. În mod caracteristic, seturile de rezultate sunt generate prin apelarea funcţiei `mysql_store_result()` sau a funcţiei `mysql_use_result()`. Totuşi, unele funcţii din biblioteca client generează seturi de rezultate în mod implicit, iar dumneavoastră sunteţi responsabil şi cu eliberarea memoriei alocate acestor seturi. Aceste funcţii sunt `mysql_list_dbs()`, `mysql_list_fields()`, `mysql_list_processes()` şi `mysql_list_tables()`.

- `char *`
  `mysql_info` `(MYSQL *conn);`

  Returnează un şir care conţine informaţii despre efectul celei mai recent executate interogări având unul din următoarele tipuri. Formatul şirului este dat imediat după fiecare interogare:

  ```
  ALTER TABLE ...
      Records: 0 Duplicates: 0 Warnings: 0
  INSERT INTO ... SELECT ...
      Records: 0 Duplicates: 0 Warnings: 0
  INSERT INTO ... VALUES (...),(...),...
      Records: 0 Duplicates: 0 Warnings: 0
  LOAD DATA ...
      Records: 0 Duplicates: 0 Warnings: 0
  UPDATE ...
      Records: 0 Duplicates: 0 Warnings: 0
  ```

  Valorile numerice vor varia în funcţie de interogarea pe care aţi executat-o, desigur.

  Pentru instrucţiunile care nu au fost afişate în lista precedentă, funcţia `mysql_info()` returnează `NULL`. Funcţia `mysql_info()` returnează valori diferite de `NULL` pentru interogarea `INSERT INTO ... VALUES` numai dacă instrucţiunea conţine mai mult de o listă de valori.

  Şirul returnat de funcţia `msyql_info()` foloseşte limbajul utilizat de server, deci nu vă puteţi baza pe posibilitatea de a-l analiza prin căutarea anumitor cuvinte.

- `my_ulonglong`
  `mysql_insert_id``(MYSQL conn);`

  Returnează valoarea `AUTO_INCREMENT` generată de interogarea cel mai recent executată pe conexiunea dată. Returnează zero dacă nu a fost executată nici o interogare sau dacă interogarea anterioară nu a generat o valoare `AUTO_INCREMENT`. Aceasta înseamnă că trebuie să apelaţi funcţia `mysql_insert_id()` imediat după o interogare la care vă aşteptaţi să genereze o valoare nouă. Dacă o altă interogare intervine între interogarea respectivă şi locul în care doriţi să utilizaţi valoarea, valoarea funcţiei `mysql_insert_id()` va fi reiniţializată de interogarea intermediară.

  O valoare returnată egală cu zero este diferită de orice valoare `AUTO_INCREMENT`, deoarece aceste valori încep de la `1`. (Excepţie: dacă creaţi o coloană `AUTO_INCREMENT` şi apoi inseraţi un număr literal negativ, secvenţa va începe de la numărul respectiv şi poate, într-un final, ajunge la `0` ca membru valabil al secvenţei. În acest caz, se presupune că ştiţi ce faceţi, mai ales deoarece funcţia `mysql_insert_id()` returnează un număr fără semn şi va trebui să „jonglaţi" cu valoarea returnată.)

  Reţineţi că funcţia `mysql_insert_id()` se comportă într-un mod diferit de funcţia SQL `LAST_INSERT_ID()`. Funcţia `mysql_insert_id()` este menţinută de client şi este stabilită pentru fiecare interogare în parte. Valoarea funcţiei `LAST_INSERT_ID()` este menţinută în server şi persistă de la o interogare la alta.

  Valorile returnate de funcţia `mysql_insert_id()` sunt specifice conexiunii şi nu sunt afectate de activitatea `AUTO_INCREMENT` pe alte conexiuni.

  Vezi nota despre afişarea valorilor de tipul `my_ulonglong` în secţiunea „Tipuri de date scalare".

- `unsigned int`
  `mysql_num_fields``(MYSQL_RES *res_set);`

  Returnează numărul coloanelor din setul de rezultate. Funcţia `mysql_num_rows()` este frecvent folosită pentru a parcurge ciclic coloanele rândului curent al setului, aşa cum se poate vedea în următorul exemplu:

  ```
  MYSQL_ROW     row;
  unsigned int  i;

  while((row = mysql_fetch_row(res_set)) != NULL)
  {
      for (i = 0; i < mysql_num_fields(res_set); i++)
      {
          /* aici se executa operatii cu randul row[i]... */
      }
  }
  ```

  Anterior versiunii MySQL 3.22.24, funcţia `mysql_num_fields()` era de asemenea folosită pentru executarea operaţiei care este acum efectuată de `mysql_field_count()`, în speţă verificarea dacă o valoare `NULL` returnată de funcţia `mysql_store_result()` sau `mysql_use_result()` indică sau nu apariţia unei erori. Acesta este motivul pentru care, în programele sursă mai vechi, veţi vedea uneori că funcţia `mysql_num_fields()` este apelată cu un pointer spre o variabilă de tratare a conexiunii, în loc de un pointer la un set de rezultate. Funcţia `mysql_num_fields()` se putea apela în ambele moduri. Utilizarea funcţiei `mysql_num_fields()` cu o variabilă de tratare a conexiunilor nu se mai practică în prezent. Pentru aceasta, trebuie să scrieţi programe care folosesc funcţia `mysql_field_count()`; descrierea funcţiei respective vă arată cum să o folosiţi chiar şi cu versiuni mai vechi ale sistemului MySQL.

- `my_ulonglong`
  `mysql_num_rows``(MYSQL_RES *res_set);`

  Returnează numărul de rânduri ale setului de rezultate. Dacă generaţi setul de rezultate cu funcţia `mysql_store_result()`, puteţi apela funcţia `mysql_num_rows()` în orice moment după aceea:

```
if ((res_set = mysql_store_result(conn) == NULL)
{
        /* acum se poate apela functia mysql_num_rows() */
}
```

Dacă generați setul de rezultate cu funcția `mysql_use_result()`, funcția `mysql_num_rows()` nu va returna valoarea corectă decât după ce ați preluat toate rândurile:

```
if ((res_set = mysql_use_result(conn)) == NULL)
{
        /* functia mysql_num_rows() nu poate fi inca apelata */
        while ((row = mysql_fetch_row(res_set)) != NULL)
        {
                /* mysql_num_rows() tot nu poate fi apelata */
        }
        /* Acum poate fi apelata functia mysql_num_rows() */
```

Vezi nota despre afișarea valorilor de tipul `my_ulonglong` în secțiunea „Tipuri de date scalare".

- `MYSQL_ROW_OFFSET`

  `mysql_row_seek``(MYSQL_RES *res_set, MYSQL_ROW_OFFSET offset);`

  Caută într-un anumit rând din setul de rezultate. Funcția `mysql_row_seek()` este similară cu `mysql_data_seek()`, dar valoarea `offset` nu este un număr de rând; `offset` trebuie să fie o valoare obținută în urma unui apel la funcția `mysql_row_tell()` sau la funcția `mysql_row_seek()`.

  Funcția `mysql_row_seek()` returnează offset-ul rândului precedent.

  Funcția `mysql_row_seek()` impune regăsirea întregului set de rezultate, deci o puteți folosi numai dacă setul de rezultate a fost creat de funcția `mysql_store_result()`, nu de funcția `mysql_use_result()`.

- `MYSQL_ROW_OFFSET`

  `mysql_row_tell``(MYSQL_RES *res_set);`

  Returnează un offset care reprezintă poziția curentă a rândului în setul de rezultate. Acesta nu este un număr de rând; valoarea poate fi transferată numai funcției `mysql_row_seek()`, nu funcției `mysql_data_seek()`.

  Funcția `mysql_row_tell()` impune regăsirea întregului set de rezultate, deci o puteți folosi numai dacă setul de rezultate a fost creat de funcția `mysql_store_result()`, nu de funcția `mysql_use_result()`.

- `MYSQL_RES *`

  `mysql_store_result``(MYSQL *conn);`

  După o interogare reușită, returnează setul de rezultate și îl stochează în client. Returnează `NULL` dacă interogarea nu returnează date sau dacă s-a produs o eroare. Când funcția `mysql_store_result()` returnează `NULL`, apelați funcția `mysql_field_count()` sau una dintre funcțiile de raportare a erorilor, pentru a determina dacă nu era de așteptat un set de rezultate sau dacă s-a produs o eroare.

  Când ați terminat lucrul cu setul de rezultate, transferați-l funcției `mysql_free_result()` pentru anularea alocării memoriei aferente acestuia.

  Vezi comparația între funcțiile `mysql_store_result()` și `mysql_use_result()` în tabelul F.7.

- `MYSQL_RES *`

  `mysql_use_result``(MYSQL *conn);`

  După o interogare reușită, inițiază o regăsire a unui set de rezultate, dar nu regăsește rânduri de date. Trebuie să apelați funcția `mysql_fetch_row()` pentru a regăsi rândurile unul câte unul. Returnează `NULL` dacă interogarea nu returnează date sau dacă s-a produs o eroare. Când funcția `mysql_use_result()` returnează `NULL`, apelați funcția `mysql_field_count()` sau una dintre funcțiile de raportare a erorilor, pentru a determina dacă nu era de așteptat un set de rezultate sau dacă s-a produs o eroare.

  Când ați terminat lucrul cu setul de rezultate, transferați-l funcției `mysql_free_result()` pentru anularea alocării memoriei aferente acestuia.

  Vezi comparația între funcțiile `mysql_store_result()` și `mysql_use_result()` în tabelul F.7.

  Atât funcția `mysql_store_result()`, cât și funcția `mysql_use_result()` sunt folosite pentru regăsirea seturilor de rezultate, dar influențează modul de utilizare a altor funcții de manipulare a setului de rezultate.

Rutine de informații

Aceste funcții furnizează informații despre client, server, versiunea protocolului și conexiunea curentă.

- `char *`

  `mysql_get_client_info``(void);`

  Returnează un șir terminat cu nul, care descrie versiunea bibliotecii client; de exemplu `"3.22.25"`.

- `char *`

  `mysql_get_host_info``(MYSQL *conn);`

  Returnează un șir terminat cu nul, care descrie conexiunea curentă; de exemplu, `"Localhost via UNIX socket"` (gazdă locală prin soclu UNIX) sau `"gazda.ta.com via TCP/IP"`.

- `unsigned int`

  `mysql_get_proto_info``(MYSQL *conn);`

  Returnează un număr care indică versiunea protocolului client-server folosită pentru conexiunea curentă.

- `char *`

  `mysql_get_server_info``(MYSQL *conn);`

  Returnează un șir terminat cu nul care descrie versiunea serverului; de exemplu, `"3.22.25-log"`. Sufixele pe care le puteți vedea după numărul versiunii serverului sunt

-log (caracteristica de consemnare în jurnal este activată), -debug (serverul rulează în mod depanare) sau -demo (serverul rulează în mod demo).

- char *

  mysql_stat(MYSQL *conn);

  Returnează un șir terminat cu nul care conține informații de stare a serverului, respectiv NULL dacă s-a produs o eroare. Formatul șirului poate suferi modificări. În prezent, este următorul:

  ```
  Uptime: 864034 Threads: 1 Questions: 32736 Slow queries: 50
  Opens: 1428 Flush tables: 1 Open tables: 61
  ```

  Aceste valori pot fi interpretate astfel:

  - Uptime este intervalul de timp (în secunde) de la pornirea serverului.
  - Threads este numărul de fire care rulează în server la un moment dat.
  - Questions este numărul de interogări pe care le-a executat serverul.
  - Slow queries este numărul de interogări care durează mai mult decât parametrul long_query_time al serverului.
  - Opens este numărul de tabele pe care le-a deschis serverul.
  - Flush tables este numărul de comenzi FLUSH, REFRESH și RELOAD executate de server.
  - Open tables este numărul de tabele deschise de server la un moment dat.

  Informațiile returnate de funcția mysql_stat() sunt similare celor raportate de comanda mysqladmin status. (De unde credeți că-și ia mysqladmin informațiile?)

- unsigned long

  mysql_thread_id(MYSQL *conn);

  Returnează numărul firului pe care serverul îl asociază conexiunii curente. Puteți folosi acest număr ca identificator pentru funcția mysql_kill().

  Se recomandă să executați funcția mysql_thread_id() chiar înainte de a avea nevoie de valoarea respectivă. Dacă obțineți valoarea și o stocați, intenționând să o folosiți mai târziu, valoarea poate fi incorectă. Aceasta se poate întâmpla în cazul în care conexiunea dumneavoastră se întrerupe și este apoi restabilită (de exemplu cu ajutorul funcției mysql_ping()), deoarece serverul va atribui un nou identificator de fir.

Rutine administrative

Funcțiile din această secțiune vă permit să controlați aspecte ale modului de operare a serverului.

- int

  mysql_kill(MYSQL *conn, unsigned long thread_id);

  Suprimă firul de server identificat prin thread_id.

  Dacă aveți privilegiul PROCESS, puteți suprima orice fir. În caz contrar, vă puteți suprima numai propriile dumneavoastră fire.

  Funcția mysql_kill() returnează zero în caz de succes, respectiv o valoare diferită de zero în caz de eșec.

- int

  mysql_refresh(MYSQL *conn, unsigned int options);

  Această funcție are un efect similar cu funcția SQL FLUSH, cu excepția faptului că puteți cere serverului să golească simultan mai multe categorii de entități. Valoarea options trebuie să fie una sau mai multe dintre valorile prezentate în tabelul F.8.

Tabelul F.8 Opțiuni pentru funcția mysql_refresh()

Valoarea opțiunii	Acțiune executată de server
REFRESH_GRANT	Reîncarcă informațiile conținute în tabelele de acordare
REFRESH_LOG	Începe jurnale generale și de actualizare noi (cele deschise la un moment dat)
REFRESH_TABLES	Închide toate tabelele deschise
REFRESH_HOSTS	Golește zona cache aferentă gazdelor
REFRESH_STATUS	Readuce variabilele de stare la valoarea zero

  Opțiunile din tabelul F.8 sunt valori ale unor biți, deci le puteți combina folosind caracterele | sau +; de exemplu REFRESH_LOG|REFRESH_TABLES sau REFRESH_LOG+REFRESH_TABLES.

  Pentru mai multe informații despre operațiile de golire, consultați descrierea instrucțiunii FLUSH din anexa D, „Referință de sintaxă SQL".

  Funcția mysql_refresh() returnează zero în caz de reușită, respectiv o valoare diferită de zero în caz de eșec.

- int

  mysql_shutdown(MYSQL *conn);

  Indică serverului să-și înceteze execuția. Pentru această operație, trebuie să dispuneți de privilegiul SHUTDOWN.

  Funcția mysql_shutdown() returnează zero în caz de reușită, respectiv o valoare diferită de zero în caz de eșec.

Rutine de depanare

Aceste funcții vă permit să generați informații de depanare la extremitatea client sau server a conexiunii. Aceasta impune ca sistemul MySQL să fie compilat astfel încât să accepte depanarea. (Utilizați opțiunea --with-debug atunci când configurați distribuția MySQL.)

- void

  mysql_debug(char *debug_string);

  Execută o operație DBUG_PUSH folosind șirul debug_string. Formatul șirului este descris în manualul de referință MySQL.

  Pentru a putea folosi funcția mysql_debug(), biblioteca client trebuie compilată cu suportul pentru depanare.

- int
  mysql_dump_debug_info(MYSQL *conn);

  Indică serverului să scrie informații de depanare în jurnal. Pentru a executa această operație, trebuie să dispuneți de privilegiul PROCESS.

  Funcția mysql_dump_debug_info()returnează zero în caz de reușită, respectiv o valoare diferită de zero în caz de eșec.

Rutine scoase din uz

Biblioteca client MySQL include un număr de funcții care în prezent sunt scoase din uz, deoarece există modalități preferate de a executa aceleași operații. Majoritatea acestor funcții pot fi înlocuite prin transferul unei interogări echivalente către funcția mysql_query(). De exemplu, apelul mysql_create_db("nume_db") se poate înlocui cu următorul apel:

```
mysql_query(conn, "CREATE DATABASE nume_db")
```

Câteva funcții, printre care mysql_connect() și mysql_eof() sunt scoase din uz deoarece au fost înlocuite de funcții care execută mai multe operații sau care furnizează o cantitate mai mare de informații.

În timp, tot mai multe funcții sunt scoase din uz, deoarece MySQL înțelege tot mai multe instrucțiuni SQL. De exemplu, când a fost adăugată instrucțiunea SQL FLUSH PRIVILEGES, funcția mysql_reload() a devenit depășită. Descrierile următoare indică versiunea sistemului MySQL când funcția respectivă a fost scoasă din uz, respectiv modalitatea preferată de execuție a fiecărei funcții în prezent. Dacă biblioteca dumneavoastră client este mai veche decât versiunea MySQL menționată, trebuie să folosiți funcția scoasă din uz, desigur.

Dacă doriți să planificați aplicații viitoare, trebuie să evitați toate funcțiile enumerate în această secțiune. Unele sau chiar toate vor dispărea în versiunea MySQL 4.0.

- MYSQL
  mysql_connect(MYSQL *conn, char *nume_gazda,
  char *nume_utilizator, char *parola);

  Aceasta este predecesoarea funcției mysql_real_connect(). De fapt, este în prezent implementată ca un apel la funcția mysql_real_connect().

  Această funcție este scoasă din uz începând cu versiunea MySQL 3.22.0.

- int
  mysql_create_db(MYSQL *conn, char *nume_db);

  Creează o bază de date cu numele dat de nume_db. Această operație este acum posibilă prin emiterea unei comenzi CREATE DATABASE cu funcția mysql_query().

  Funcția mysql_create_db() returnează zero în caz de reușită, respectiv o valoare diferită de zero în caz de eșec.

  Această funcție este scoasă din uz începând cu versiunea MySQL 3.21.15.

- int
  mysql_drop_db(MYSQL *conn, char *nume_db);

  Șterge baza de date specificată de nume_db. În prezent, această operație se poate executa prin emiterea unei instrucțiuni DROP DATABASE cu funcția mysql_query().

  Funcția mysql_drop_db() returnează zero în caz de reușită, respectiv o valoare diferită de zero în caz de eșec.

  Această funcție este scoasă din uz începând cu versiunea MySQL 3.21.15.

- my_bool
  mysql_eof(MYSQL_RES *res_set);

  Returnează o valoare diferită de zero dacă s-a obținut un set de rezultate, respectiv zero dacă s-a produs vreo eroare. Funcția mysql_eof() este folosită când utilizați combinația între funcțiile mysql_use_result() pentru a iniția o operație de regăsire a unui set de rezultate și mysql_fetch_row() pentru a regăsi rândurile de date unul câte unul. Cu funcția mysql_use_result(), o valoare NULL returnată de funcția mysql_fetch_row() poate însemna fie că s-a ajuns la sfârșitul setului de rezultate, fie apariția unei erori. Funcția mysql_eof() sesizează diferența între cele două rezultate.

  În prezent, pentru a obține același rezultat se pot folosi funcțiile mysql_errno() și mysql_error(), deși cele două funcții returnează o cantitate mai mare de informații. (Funcțiile indică și motivul producerii oricărei erori, nu se rezumă la a semnala pur și simplu producerea erorii.)

  Această funcție a fost scoasă din uz începând cu versiunea MySQL 3.21.17.

- MYSQL_RES *
  mysql_list_dbs(MYSQL *conn, char *wild);

  Returnează un set de rezultate care enumeră numele bazelor de date din server, respectiv NULL dacă s-a produs o eroare. Această listă conține toate bazele de date care corespund expresiei regulate SQL indicate de wild (care poate conține caracterele de înlocuire % și _) sau toate bazele de date, dacă wild este NULL. Dumneavoastră sunteți responsabil cu apelarea funcției mysql_free_result() pentru anularea alocării memoriei aferente setului de rezultate.

  Lista produsă de funcția mysql_list_dbs() poate fi obținută prin executarea unei comenzi SHOW DATABASES cu mysql_query() și apoi prin prelucrarea setului de rezultate.

  Această funcție a fost scoasă din uz începând cu versiunea MySQL 3.22.0.

- MYSQL_RES *
  mysql_list_fields(MYSQL *conn, char *nume_tbl, char *wild);

  Returnează un set de rezultate care enumeră numele coloanelor din tabelul dat, respectiv NULL dacă se produce vreo eroare. Lista conține numele tuturor coloanelor care corespund expresiei regulate SQL indicate de wild (care poate conține caracterele de înlocuire % și _) sau toate coloanele, dacă wild este NULL. Dumneavoastră sunteți responsabil cu apelarea funcției mysql_free_result() pentru anularea alocării memoriei aferente setului de rezultate.

Lista produsă de funcția `mysql_list_fields()` poate fi obținută prin executarea unei comenzi `SHOW COLUMNS` cu `mysql_query()` și apoi prin prelucrarea setului de rezultate.

Această funcție a fost scoasă din uz începând cu versiunea MySQL 3.22.0.

- `MYSQL_RES *`

  `mysql_list_processes``(MYSQL *conn);`

  Returnează un set de rezultate care conține o listă a proceselor care rulează pe server, respectiv `NULL` dacă s-a produs o eroare. Dacă dispuneți de privilegiul `PROCESS`, lista conține toate procesele server. Dacă nu, lista conține propriile dumneavoastră procese. Dumneavoastră sunteți responsabil cu apelarea funcției `mysql_free_result()` pentru anularea alocării memoriei aferente setului de rezultate.

  Lista produsă de funcția `mysql_list_fields()` poate fi obținută prin executarea unei comenzi `SHOW PROCESSLIST` cu `mysql_query()` și apoi prin prelucrarea setului de rezultate.

  Această funcție a fost scoasă din uz începând cu versiunea MySQL 3.22.0.

- `MYSQL_RES *`

  `mysql_list_tables``(MYSQL *conn, char *wild);`

  Returnează un set de rezultate care enumeră numele tabelelor din baza de date curentă, respectiv `NULL` dacă se produce vreo eroare. Lista conține numele tuturor coloanelor care corespund expresiei regulate SQL indicate de `wild` (care poate conține caracterele de înlocuire `%` și `_`) sau toate tabelele, dacă `wild` este `NULL`. Dumneavoastră sunteți responsabil cu apelarea funcției `mysql_free_result()` pentru anularea alocării memoriei aferente setului de rezultate.

  Lista produsă de funcția `mysql_list_tables()` poate fi obținută prin executarea unei comenzi `SHOW TABLES` cu `mysql_query()` și apoi prin prelucrarea setului de rezultate.

  Această funcție a fost scoasă din uz începând cu versiunea MySQL 3.22.0.

- `int`

  `mysql_reload``(MYSQL *conn);`

  Indică serverului să reîncarce tabelele de acordare. Această operație se poate executa în prezent prin emiterea unei interogări `FLUSH PRIVILEGES` cu funcția `mysql_query()`. Pentru a folosi funcția `mysql_reload()`, trebuie să dispuneți de privilegiul `RELOAD`.

  Funcția `mysql_reload()` returnează zero în caz de reușită, respectiv o valoare diferită de zero în caz de eșec.

  Această funcție a fost scoasă din uz începând cu versiunea MySQL 3.21.9.

ANEXA G

Referință API Perl DBI

Această anexă descrie interfața de programare a aplicațiilor Perl DBI. Interfața constă dintr-un set de metode și atribute pentru comunicarea cu serverele de baze de date și pentru accesul la bazele de date din scripturile Perl. De asemenea, anexa descrie extensiile la DBI specifice sistemului MySQL furnizate de DBD::mysql, driverul MySQL pentru baze de date.

Unele metode și atribute DBI nu vor fi discutate aici, fie deoarece nu se aplică sistemului MySQL, fie deoarece sunt metode noi, experimentale, care se pot modifica în timpul dezvoltării sau care au fost chiar abandonate. Unele metode DBD specifice sistemului MySQL nu au fost discutate, deoarece sunt depășite. Dacă doriți mai multe informații despre metodele noi sau despre cele depășite, consultați documentația DBI sau documentația driverului DBD pentru MySQL, pe care o puteți obține prin rularea următoarelor comenzi:

```
% perldoc DBI
% perldoc DBI::FAQ
% perldoc DBD::mysql
```

Această anexă servește drept referință, deci include numai scurte fragmente de program care ilustrează utilizarea interfeței API pentru Perl DBI. Pentru scripturi client complete și note privind redactarea lor, consultați capitolul 7, „Interfața API pentru Perl DBI".

Scrierea scripturilor

Fiecare script Perl care folosește modulul DBI trebuie să includă următoarea linie:

```
use DBI;
```

Nu este necesară includerea unei linii `use` pentru un anumit modul de nivel DBD, deoarece DBI va activa modulul adecvat atunci când vă conectați la server.

În mod normal, un script DBI va deschide o conexiune folosind metoda `connect()`, va emite interogări și apoi va închide conexiunea cu metoda `disconnect()`. Metodele folosite pentru emiterea interogărilor variază. De regulă, interogările non-`SELECT` sunt executate prin transferul interogării către funcția `prepare()`, apoi apelarea metodei `execute()` și în final regăsirea rând cu rând a rezultatelor interogării într-un ciclu care invocă în mod repetat o metodă de preluare a rândurilor, precum `fetchrow_array()` sau `fetchrow_hashref()`.

Metode DBI

Descrierile metodelor prezentate aici sunt scrise într-un format oarecum diferit de cel folosit pentru funcțiile C din Anexa F, „Referință API C", respectiv de formatul funcțiilor PHP care va fi prezentat în Anexa H, „Referință API PHP". Funcțiile din aceste anexe sunt scrise în formă de prototip, enumerând în mod explicit tipurile valorilor returnate și tipurile de parametri. În general, Perl nu folosește prototipuri în acest mod, deci descrierile din această anexă indică tipurile valorilor returnate și tipurile parametrilor ca variabile, unde caracterul inițial al variabilei indică tipul valorii: `$` pentru un scalar, `@` pentru un tablou și `%` pentru o valoare hash (tablou asociativ). În plus, pentru orice parametru care începe cu un caracter `\`, este necesar transferul unei referințe la variabila respectivă, nu al variabilei în sine. Un nume de variabilă care se termină cu particula `_ref` arată că valoarea variabilei este o referință.

În această anexă se vor folosi numeroase variabile cu semnificații specifice. Aceste variabile sunt prezentate în tabelul G.1.

Tabelul G.1 Nume convenționale de variabile Perl

Nume	Semnificație
`$drh`	Variabilă de manipulare a unui obiect driver
`$dbh`	Variabilă de manipulare a unui obiect bază de date
`$sth`	Variabilă de manipulare a unui obiect instrucțiune (interogare)
`$fh`	Variabilă de manipulare a unui fișier deschis
`$h`	Variabilă de manipulare „generală"; semnificația depinde de conținut
`$rc`	Codul returnat de operații care returnează „adevărat" sau „fals"
`$rv`	Valoarea returnată de operații care returnează un întreg
`$rows`	Valoarea returnată de operații care returnează un număr de rânduri
`@ary`	Un tablou (listă) care reprezintă un rând de valori returnate de o interogare

Multe metode acceptă un argument hash `%attr`, care conține atribute ce influențează modul de funcționare a metodei. Acest argument hash trebuie transferat prin referință. Puteți face aceasta în două moduri. O modalitate este stabilirea conținutului valorii hash `%attr` înainte de a invoca metoda, iar apoi transferul acesteia către metodă:

```
my (%attr) = (AttrName1 => valoare1, AttrName2 => valoare2);
$ret_val = $h->method(..., \%attr);
```

Cealaltă metodă constă în furnizarea unui hash anonim direct în declarația de invocare a metodei:

```
$ret_val = $h->method
    (...,{AttrName1 => valoare1, AttrName2 => valoare2});
```

Informațiile opționale sunt indicate prin paranteze drepte ([ ]). Modul de utilizare a unei metode sau a unei funcții este indicat de către secvența apelantă. `DBI->` indică o metodă de clasă DBI, `DBI::` indică o funcție DBI, iar `$DBI::` indică o variabilă DBI. Pentru metodele care sunt apelate folosind variabile de manipulare, numele variabilei de manipulare indică aria de acțiune a metodei. `$dbh->` indică o metodă de manipulare a unei baze de date, `$sth->` indică o metodă de manipulare a instrucțiunilor, iar `$h->` indică o metodă care poate fi apelată cu diferite tipuri de variabile de manipulare. Iată un exemplu de secvență apelantă:


```
@row_ary = $dbh->selectrow_array ($instructiune,
             [\%attr [, @valori_asociere]]);
```

Această secvență arată că metoda `selectrow_array()` a fost apelată ca metodă de manipulare a bazelor de date, deoarece este invocată folosind `$dbh->`. Parametrii sunt `$instructiune` (o valoare scalară), `%attr` (o valoare hash care trebuie transferată ca referință, așa cum o arată caracterul `\` de la început) și `@valori_asociere` (un tablou). Al doilea și al treilea parametru sunt opționali. Valoarea returnată este un tablou.

Descrierile metodelor indică valoarea returnată în caz de eroare, dar o eroare poate fi returnată numai dacă atributul `RaiseError` este dezactivat. Dacă acest atribut este activat, scriptul își va încheia execuția în mod automat.

În descrierile care urmează, termenul „interogare `SELECT`" va desemna atât o interogare `SELECT`, cât și orice altă interogare care returnează rânduri, cum ar fi `DESCRIBE`, `EXPLAIN` sau `SHOW`.

Metodele de clasă DBI

Parametrul `%attr` pentru metodele din această secțiune se poate folosi pentru a specifica atribute de prelucrare a metodelor. Pentru MySQL, cele mai importante atribute sunt `PrintError` și `RaiseError`. De exemplu, pentru a activa terminarea automată a execuției scriptului la apariția unei erori DBI, activați atributul `RaiseError`:

```
$dbh = DBI->connect($sursa_date, $nume_utilizator, $parola,
    {RaiseError => 1});
```

Despre atributele `PrintError` și `RaiseError` vom discuta în secțiunea „Atribute generale ale variabilelor manipulare".

- `@ary = DBI->`**`available_drivers`**`([$quiet]);`

  Returnează o listă a driverelor DBI disponibile. Dacă sunt găsite mai multe drivere cu același nume, va fi emis un mesaj de avertizare. Puteți suprima acest mesaj transferând o valoare egală cu 1 a parametrului `$quiet`.

- `$dbh = DBI->`**`connect`**`($sursa_date, $nume_utilizator, $parola [, \%attr]);`

  Metoda `connect()` stabilește o conexiune cu serverul de baze de date și returnează o variabilă de manipulare pentru baze de date. Variabila este `undef` dacă încercarea de conexiune eșuează. Pentru a termina o conexiune stabilită cu succes, invocați metoda `disconnect()` folosind variabila de manipulare pentru baze de date returnată de metoda `connect()`.

  Pentru metoda `connect()`, toate atributele specificate în parametrul `%attr` influențează global prelucrarea metodelor DBI.

  Sursa de date poate fi dată în mai multe forme. Prima parte este întotdeauna `DBI:mysql`, unde `DBI` poate fi scris atât cu majuscule, cât și cu minuscule, iar numele driverului, în speță `mysql`, trebuie scris cu minuscule. Toate informațiile scrise în sursa

de date după numele driverului sunt interpretate de driver, deci sintaxa descrisă în expunerea următoare nu se aplică în mod obligatoriu oricărui alt driver decât DBD::mysql.

După numele driverului, mai puteți specifica un nume al bazei de date și un nume de gazdă în partea inițială a șirului care reprezintă sursa de date:

```
$sursa_date = "DBI:mysql:nume_db";
$sursa_date = "DBI:mysql:nume_db:nume_gazda";
```

Baza de date poate fi specificată sub forma `nume_db` sau `database=nume_db`. Numele gazdei poate fi specificat sub forma `nume_gazda` sau `host=nume_gazda`.

După partea de început a șirului care reprezintă sursa de date, puteți specifica mai multe opțiuni folosind formatul `atribut=valoare`. Fiecare opțiune trebuie precedată printr-un caracter punct și virgulă. Driverul MySQL acceptă următoarele opțiuni:

- `host=nume_gazda`

  Gazda la care se va stabili conexiunea. Se mai poate specifica și un număr de port, folosind formatul `nume_gazda:numar_port`.

- `port=numar_port`

  Numărul portului la care se va stabili conexiunea.

- `mysql_compression = 1`

  Această opțiune activează protocolul client-server de comunicare comprimată între client și serverul MySQL.

  Opțiunea `mysql_compression` necesită versiunea MySQL 3.22.3 sau o versiune ulterioară, respectiv versiunea DBD::mysql 1.19.20 sau o versiune ulterioară.

- `mysql_read_default_file=nume_fisier`

  În mod prestabilit, scripturile DBI nu citesc fișiere cu opțiuni MySQL pentru a obține parametrii de conexiune. Această opțiune vă permite să specificați un fișier de opțiune care va fi citit. Numele fișierului trebuie să reprezinte calea completă de acces la acesta. (În caz contrar, va fi interpretat relativ la catalogul unde executați scriptul, iar dumneavoastră veți obține rezultate inconsecvente.)

  Pentru a citi fișierul `.my.cnf` din catalogul de bază al oricărui utilizator care se întâmplă să ruleze scriptul, specificați numele fișierului sub forma `$ENV{HOME}/.my.cnf`. Această notație este utilă când un script va fi utilizat de mai multe persoane și dumneavoastră doriți ca acestea să se conecteze sub propriile conturi, nu sub contul unui utilizator pe care îl codați hard în script.

  Opțiunea `mysql_read_default_file` necesită MySQL versiunea 3.22.10 sau o versiune ulterioară, respectiv versiunea DBD::mysql 1.21.06 sau o versiune ulterioară.

- `mysql_read_default_group=nume_grup`

  În cazul în care citiți un fișier cu opțiuni pentru a determina parametrii de conexiune, grupul `[client]` este citit în mod prestabilit. Opțiunea `mysql_read_default_group` vă permite să specificați un grup care va fi citit în afară de grupul `[client]`. De exemplu, `mysql_read_default_group=dbi` specifică faptul că, în afara grupului `[client]`, mai trebuie citit și grupul `[dbi]`. Precedența parametrilor care se găsesc în mai multe grupuri este descrisă în anexa E, „Referință de programe MySQL".

  Opțiunea `mysql_read_default_group` necesită MySQL versiunea 3.22.10 sau o versiune ulterioară, respectiv versiunea DBD::mysql 1.21.06 sau o versiune ulterioară.

- `mysql_socket=nume_soclu`

  Această opțiune specifică numele căii de acces la soclul de domeniu UNIX care va fi utilizat pentru conexiunile cu `localhost`, în cazul în care calea spre soclul local nu este convenabilă.

  Opțiunea `mysql_socket` necesită MySQL 3.21.15 sau o versiune ulterioară. DBI consultă mai multe variabile de mediu pentru a determina parametrii de conexiune:

  - Dacă sursa de date este nedefinită sau vidă, se va folosi valoarea variabilei `DBI_DSN`.
  - Dacă numele driverului lipsește din sursa de date, se va folosi valoarea variabilei `DBI_DRIVER`.
  - Dacă parametrii `nume_utilizator` sau `parola` ai apelului la metoda `connect()` nu sunt definiți, se vor folosi valorile variabilelor `DBI_USER` și `DBI_PASS`. Acest fapt *nu* se întâmplă dacă parametrii sunt alcătuiți din șiruri vide.

  Utilizarea variabilei `DBI_PASS` reprezintă un pericol la adresa securității, deci nu trebuie să o folosiți în nici o situație în care securitatea este importantă. (Valorile variabilelor de mediu sunt vizibile altor utilizatori prin intermediul unor comenzi precum ps.)

  Dacă unii parametri de conexiune rămân necunoscuți după consultarea tuturor surselor de informații, DBI folosește valorile prestabilite. Dacă numele gazdei nu este specificat, se va folosi numele prestabilit `localhost`. Dacă numele utilizatorului nu este specificat, acesta va avea ca valoare prestabilită numele dumneavoastră de deschidere a sesiunii de lucru (sub UNIX), respectiv `ODBC` (sub Windows). Dacă nu este specificată parola, nu există nici o valoare prestabilită. Nu va fi trimisă nici o parolă.

- `$drh = DBI->`**`install_driver`**`($driver_name)`

  Activează un driver de nivel DBD și returnează o variabilă de manipulare a acestuia. Pentru MySQL, numele driverului este `"mysql"` (trebuie scris cu minuscule). În mod normal, utilizarea acestei metode nu este necesară, deoarece DBI activează în mod automat driverul adecvat atunci când invocați metoda `connect()`. Totuși, metoda `install_driver()` poate fi utilă dacă folosiți metoda `func()` pentru a efectua operații administrative. (Vezi secțiunea „Metode administrative specifice sistemului MySQL".)

Metode de manipulare a bazelor de date

Metodele din această secțiune sunt invocate cu ajutorul unei variabile de manipulare a bazelor de date și se pot folosi după ce ați obținut o atare variabilă prin apelarea metodei `connect()`.

Parametrul `%attr` pentru metodele din această secțiune se poate folosi pentru a specifica atribute de prelucrare a metodelor. Pentru MySQL, cele mai importante din acestea sunt `PrintError` și `RaiseError`. De exemplu, puteți activa `RaiseError` pentru a determina încheierea automată a execuției scriptului în cazul producerii unei erori DBI în timpul prelucrării unei anumite interogări, ca aceasta:

```
$rows = $dbh-do($instructiune, {Raise_Error = 1});
```

Despre atributele `PrintError` și `RaiseError` vom discuta în secțiunea „Atribute generale de manipulare".

- `$rc = $dbh->disconnect();`

  Termină conexiunea asociată variabilei de manipulare. Dacă o conexiune rămâne activă când un script își încheie execuția, este afișat un mesaj de avertizare și conexiunea este terminată automat.

- `$rows = $dbh->do($instructiune [, \%attr [, @valori_asociere]]);`

  Pregătește și execută interogarea indicată de `$instructiune`. Valoarea returnată reprezintă numărul de rânduri afectate, respectiv -1 dacă numărul de rânduri este necunoscut sau `undef` dacă se produce o eroare. Dacă numărul de rânduri afectate este zero, valoarea returnată este șirul `0E0`, care este echivalent cu zero, dar este considerat adevărat.

  Metoda `do()` este folosită mai ales pentru instrucțiuni care nu regăsesc rânduri, precum `DELETE`, `INSERT`, `REPLACE` și `UPDATE`. Dacă încercați să folosiți această metodă pentru o instrucțiune `SELECT`, nu veți primi o variabilă de manipulare a instrucțiunilor și nu veți putea prelua rânduri.

  Parametrul `@valori_asociere` este folosit când instrucțiunea conține câmpuri de înlocuire (indicate prin caracterul `?` plasat în șirul de interogare). `@valori_asociere` este o listă a valorilor care vor fi asociate câmpurilor de înlocuire. Este necesar ca numărul valorilor să fie egal cu acela al câmpurilor de înlocuire. Dacă specificați valorile care trebuie asociate, dar fără atribute, transferați `undef` ca valoare a parametrului `\%attr`.

- `$rc = $dbh->ping();`

  Verifică dacă mai este activă conexiunea cu serverul, după care returnează `true` sau `false` în consecință.

- `$sth = $dbh->prepare($instructiune [, \%attr]);`

  Pregătește interogarea indicată de `$instructiune` în vederea unei execuții ulterioare și returnează o variabilă de manipulare a instrucțiunii. Variabila de manipulare returnată se poate folosi cu metoda `execute()` pentru executarea interogării.

- `$str = $dbh->quote($valoare [, $tip_date]);`

  Prelucrează un șir pentru a efectua încadrarea între ghilimele și modificarea semnificației unor caractere speciale din instrucțiunile SQL, astfel încât șirul să nu cauzeze o eroare de sintaxă atunci când executați instrucțiunea. De exemplu, șirul `"I'm happy"` este returnat sub forma `"'I\'m happy'"` (fără ghilimelele duble). Dacă `$valoare` are valoarea `undef`, este returnată sub forma șirului `"NULL"` (fără semnele citării).

  De obicei, parametrul `$tip_date` nu este necesar, deoarece MySQL convertește în mod automat în alte tipuri valorile specificate ca șiruri în cadrul interogărilor. `$tip_date` poate fi specificat ca un indiciu privind tipul unei anumite valori; de exemplu, `DBI::SQL_INTEGER` arată că `$valoare` reprezintă un întreg.

  Nu folosiți metoda `quote()` cu valori pe care le veți insera într-o interogare folosind caractere de înlocuire; DBI încadrează automat aceste valori între ghilimele.

- `$ary_ref = $dbh->selectall_arrayref ($instructiune [, \%attr [, @valori_asociere]]);`

  Execută interogarea specificată de `$instructiune` și returnează rezultatul prin combinarea metodelor `prepare()`, `execute()` și `fetchall_arrayref()`. Returnează `undef` dacă se produce o eroare.

  Dacă parametrul `$instructiune` este o instrucțiune pregătită anterior, etapa `prepare()` este omisă.

  Parametrul `@valori_asociere` are aceeași semnificație ca pentru metoda `do()`.

- `$ary_ref = $dbh->selectcol_arrayref ($instructiune [\%attr [, @valori_asociere]]);`

  Execută interogarea specificată în `$instructiune` și returnează prima coloană a rezultatului prin combinarea metodelor `prepare()` și `execute()`. Rezultatul este returnat ca o referință la un tablou care conține prima coloană din fiecare rând. Dacă se produce o eroare, metoda returnează `undef`.

  Dacă parametrul `$instructiune` este o instrucțiune pregătită anterior, etapa `prepare()` este omisă.

  Parametrul `@valori_asociere` are aceeași semnificație ca pentru metoda `do()`.

- `$row_ary = $dbh->selectrow_array ($instructiune [, \%attr [, @valori_asociere]]);`

  Execută interogarea specificată în `$instructiune` și returnează primul rând al rezultatului, prin combinarea metodelor `prepare()`, `execute()` și `fetchrow_array()`.

  Dacă parametrul `$instructiune` este o instrucțiune pregătită anterior, etapa `prepare()` este omisă.

  Când este apelată într-un context de listă, metoda `selectrow_array()` returnează un tablou care reprezintă valorile din rând, respectiv un tablou vid dacă se produce o eroare. În context scalar, metoda `selectrow_array()` returnează valoarea primului element din tablou (prima coloană a rândului). Metoda returnează `undef` în cazul producerii unei erori.

  Parametrul `@valori_asociere` are aceeași semnificație ca pentru metoda `do()`.

Metode de manipulare a instrucțiunilor

Metodele din această secțiune sunt invocate prin intermediul unei variabile de tratare a instrucțiunilor, pe care o obțineți prin apelarea metodei `prepare()`.

- `$rc = $sth->bind_col($numar_coloana, \$var_de_asociat);`

  Asociază o coloană dată pentru o interogare `SELECT` la o variabilă Perl, care trebuie transferată ca referință. `$numar_coloana` trebuie să se găsească în intervalul cuprins între 1 și numărul coloanelor selectate de interogare. La fiecare preluare a unui rând, variabila este actualizată automat cu valoarea din coloană.

  Metoda `bind_col()` trebuie apelată după `prepare()` și înainte de `execute()`.

  Metoda `bind_col()` returnează „fals" dacă numărul coloanei nu se află în domeniul cuprins între 1 și numărul coloanelor selectate de interogare.

- `$rc = $sth->bind_columns(\$var_de_asociat1,\$var_de_asociat2, ...)`

  Asociază o listă cu variabile coloanelor returnate de o instrucţiune `SELECT` pregătită. Examinaţi descrierea metodei `bind_col()`.

  Metoda `bind_columns()` returnează „fals" dacă numărul de referinţe nu corespunde numărului coloanelor selectate de interogare.

- `$rv = $sth->bind_param($n, $valoare [, \%attr]);`

  `$rv = $sth->bind_param($n, $valoare [, $tip_asociere]);`

  Asociază o valoare cu un câmp de înlocuire (?) dintr-o instrucţiune. Metoda trebuie apelată după `prepare()` şi înainte de `execute()`.

  `$n` specifică numărul câmpului de înlocuire la care trebuie asociată valoarea `$valoare` şi trebuie să fie cuprins între 1 şi numărul câmpurilor de înlocuire. Pentru a asocia valoarea `NULL`, transferaţi `undef`.

  Parametrul `\%attr` sau `$tip_asociere` poate fi furnizat ca indiciu privind tipul valorii care va fi asociată. De exemplu, pentru a arăta că valoarea reprezintă un întreg, puteţi invoca metoda `bind_param()` în oricare din următoarele moduri:

```
$rv = $sth->bind_param($n, $valoare , { TYPE => DBI::SQL_INTEGER });
$rv = $sth->bind_param($n, $valoare , DBI::SQL_INTEGER);
```

  În mod prestabilit, variabila va fi tratată ca `VARCHAR`, ceea ce este suficient în condiţii normale, deoarece MySQL converteşte valorile de tip şir din interogări în alte tipuri de date, conform necesităţilor.

- `$rows = $sth->dump_results([$lungmax [, $sep_linie [, $sep_camp[, $fh]]]]);`

  Preia toate rândurile de la variabila de manipulare a instrucţiunilor `$sth`, le formatează apelând funcţia utilitară `DBI::neat_list()` şi le afişează în variabila de manipulare a fişierelor dată. Returnează numărul rândurilor preluate.

  Valorile prestabilite pentru parametrii `$lungmax`, `$sep_linie`, `$sep_camp` şi `$fh` sunt 35, `"\n"`, `", "` şi `STDOUT`.

- `$rv = $sth->execute([@valori_asociere]);`

  Execută o instrucţiune pregătită. Returnează „adevărat" dacă instrucţiunea a fost executată cu succes, respectiv `undef` dacă s-a produs o eroare.

  Parametrul `@valori_asociere` are aceeaşi semnificaţie ca în cazul metodei `do()`.

- `$ary_ref = $sth->fetch();`

  `fetch()` este un alias pentru `fetchrow_arrayref()`.

- `$tbl_ary_ref = $sth->fetchall_arrayref([$slice_array_ref]);`

  `$tbl_ary_ref = $sth->fetchall_arrayref([$slice_hash_ref]);`

  Preia toate rândurile de la variabila de manipulare a instrucţiunii `$sth` şi returnează o referinţă la un tablou care conţine o referinţă pentru fiecare rând preluat. Semnificaţia fiecărei referinţe din tablou depinde de argumentul pe care îl transferaţi. Dacă nu este dat nici un argument sau este dat un argument referinţă la o „felie" de tablou, fiecare element al valorii `$tbl_ary_ref` este o referinţă la un tablou care conţine valorile unui rând al rezultatului. Cu un argument referinţă la o „felie" a unei valori hash, fiecare element al valorii `$tbl_ary_ref` este o referinţă la o valoare hash care conţine valorile unui rând al rezultatului.

- `@ary = $sth->fetchrow_array();`

  Când este apelată într-un context listă, metoda `fetchrow_array()` returnează un tablou care conţine valorile coloanelor din următorul rând al setului de rezultate, respectiv un tablou vid dacă nu mai există rânduri sau dacă s-a produs o eroare. Într-un context scalar, metoda `fetchrow_array()` returnează valoarea primului element al tabloului (adică prima coloană a rândului), respectiv `undef` dacă nu mai sunt rânduri sau dacă s-a produs o eroare.

  Puteţi face diferenţa între etalarea normală a setului de rezultate şi o eroare prin verificarea valorii variabilei `$sth-err()`. O valoare egală cu zero arată că aţi ajuns la sfârşitul setului de rezultate, fără erori.

- `$ary_ref = $sth->fetchrow_arrayref();`

  Returnează o referinţă la un tablou care conţine valori din coloane aferente următorului rând al setului de rezultate. Returnează `undef` dacă nu mai sunt rânduri sau dacă se produce o eroare.

  Puteţi face diferenţa între etalarea normală a unui set de rezultate şi o eroare prin verificarea metodei `$sth-err()`. O valoare egală cu zero arată că aţi ajuns la sfârşitul setului de rezultate, fără erori.

- `$hash_ref = $sth->fetchrow_hashref([$nume]);`

  Returnează o referinţă la o valoare hash care conţine valorile coloanelor aferente următorului rând al setului de rezultate. Dacă nu mai sunt rânduri sau dacă se produce o eroare, returnează `undef`. Valorile hash de index sunt numele coloanelor, iar elementele valorii hash sunt valorile coloanelor.

  Argumentul `$nume` poate fi specificat pentru a indica atributul variabilă de manipulare a instrucţiunilor care se va folosi pentru valorile cheilor hash. Valoarea sa prestabilită este `"NAME"`. Acest fapt poate cauza probleme, deoarece numele coloanelor nu sunt sensibile la diferenţa între majuscule şi minuscule în interogări, dar cheile hash sunt. Pentru a forţa scrierea cheilor hash cu minuscule sau cu majuscule, puteţi specifica o valoare `$nume` egală cu `"NAME_lc"` sau `"NAME_uc"`.

  Puteţi face diferenţa între etalarea normală a unui set de rezultate şi o eroare prin verificarea valorii variabilei `$sth-err()`. O valoare egală cu zero arată că aţi ajuns la sfârşitul setului de rezultate, fără erori.

- `$rc = $sth->finish();`

  Eliberează toate resursele alocate variabilei de manipulare a instrucţiunii. În mod normal, nu este necesară invocarea în mod explicit a acestei metode, dar dacă preluaţi numai o parte a unui set de rezultate, prin apelarea metodei `finish()` indicaţi DBI că aţi terminat preluarea datelor.

  Apelarea metodei `finish()` poate invalida atributele instrucţiunii; cel mai bine este să obţineţi accesul la acestea imediat după ce aţi invocat metoda `execute()`.

- `$rv = $sth->rows();`

  Returnează numărul rândurilor afectate de instrucțiunea asociată cu `$sth`, respectiv -1 dacă s-a produs o eroare. Această metodă este folosită mai ales pentru instrucțiuni care nu returnează rânduri. Pentru instrucțiunile `SELECT`, nu trebuie să contați pe metoda `rows()`; numărați rândurile pe măsură ce le preluați.

Metode generale de manipulare

Metodele din această secțiune nu sunt specifice unor anumite tipuri de variabile de manipulare. Metodele pot fi invocate folosind variabile de manipulare pentru drivere, baze de date sau instrucțiuni.

- `$h->err();`

  Returnează codul numeric al erorii pentru cel mai recent invocată operație cu driverul. Un cod zero indică absența oricărei erori.

- `$h->errstr();`

  Returnează mesajul care conține șirul de eroare pentru cel mai recent invocată operație cu driverul. Un șir vid indică absența oricărei erori.

- `DBI->trace($trace_level [, $trace_filename]);`

  `$h->trace($trace_level [, $trace_filename]);`

  Stabilește un nivel de urmărire. Urmărirea furnizează informații despre modul de operare a modulului DBI. Nivelul de urmărire poate varia între 0 (inactiv) și 9 (cantitate maximă de informații). Urmărirea poate fi activată pentru toate operațiile DBI din cadrul unui script prin invocarea funcției `trace()` ca o metodă de clasă DBI sau pentru o variabilă de manipulare individuală:

`DBI->trace(2);`	Activează urmărirea scriptului
`$sth->trace(2);`	Activează urmărirea variabilei de manipulare

  De asemenea, urmărirea mai poate fi activată la nivel global pentru toate scripturile DBI pe care le rulați, prin configurarea variabilei de mediu `DBI_TRACE`.

  Datele de ieșire obținute prin urmărire ajung în mod prestabilit la `STDERR`. Parametrul `$trace_filename` poate fi furnizat pentru a orienta datele de ieșire spre un alt fișier. Datele de ieșire vor fi atașate la conținutul existent al fișierului, indiferent de natura acestuia.

  Fiecare apel la funcția `trace()` determină orientarea spre același fișier a *tuturor* variabilelor de manipulare urmărite. Dacă nu este denumit nici un fișier, toate datele de ieșire obținute prin urmărire ajung la `STDERR`.

- `DBI->trace_msg($str [, $min_level]);`

  `$h->trace_msg($str [, $min_level]);`

  Scrie un mesaj în datele de ieșire obținute prin urmărire, dacă variabila de manipulare este urmărită sau dacă a fost activată urmărirea la nivelul modulului DBI. Metoda `trace_msg()` poate fi apelată sub forma `DBI-trace_msg()` pentru a scrie un mesaj dacă este activată urmărirea la nivelul modulului DBI.

  Parametrul `$min_level` poate fi precizat pentru a specifica faptul că mesajul trebuie scris numai dacă nivelul de urmărire se află cel puțin la valoarea respectivă.

Metode administrative specifice sistemului MySQL

Această secțiune descrie metoda `func()` furnizată de DBI ca mijloc de acces direct la operațiile specifice driverului.

```
$rc = $drh->func('createdb',
    $nume_db, $nume_gazda, $nume_utilizator, $parola, 'admin');
$rc = $drh->func('dropdb',
    $nume_db, $nume_gazda, $nume_utilizator, $parola, 'admin');
$rc = $drh->func('shutdown',
    $nume_gazda, $nume_utilizator, $parola, 'admin');
$rc = $drh->func('reload',
    $nume_gazda, $nume_utilizator, $parola, 'admin');

$rc = $dbh->func('createdb', $nume_db, 'admin');
$rc = $dbh->func('dropdb', $nume_db, 'admin');
$rc = $dbh->func('shutdown', 'admin');
$rc = $dbh->func('reload', 'admin');
```

Metoda `func()` este accesibilă prin intermediul unei variabile de manipulare a driverului sau al unei variabile de manipulare a bazelor de date. O variabilă de manipulare a driverului nu este asociată cu o conexiune deschisă, deci, dacă obțineți acces la metoda `func()` în acest mod, trebuie să furnizați argumente pentru numele gazdei, numele utilizatorului și parolă, pentru a permite metodei să stabilească o conexiune. Dacă obțineți acces la metoda `func()`cu o variabilă de manipulare a bazelor de date, aceste argumente nu mai sunt necesare. O variabilă de manipulare a driverului poate fi obținută, dacă este necesar, după cum urmează:

```
$drh = DBI->install_driver("mysql");    # ("mysql" trebuie
                                        # scris cu minuscule)
```

- `createdb`

  Creează baza de date specificată de `$nume_db`. Pentru a executa această operație, trebuie să dispuneți de privilegiul `CREATE` pentru baza de date.

- `dropdb`

  Șterge (elimină) baza de date specificată de `$nume_db`. Pentru a executa această operație, trebuie să dispuneți de privilegiul `DROP` pentru baza de date.

  Fiți atent: dacă ștergeți o bază de date, s-a terminat cu ea. Nu o puteți aduce înapoi.

- `shutdown`

  Oprește serverul. Pentru a executa această operație, trebuie să dispuneți de privilegiul `SHUTDOWN`.

- `reload`

  Indică serverului să reîncarce tabelele de acordare. Această operație este necesară dacă modificați în mod direct conținutul tabelelor de acordare, folosind instrucțiunile

DELETE, INSERT sau UPDATE în detrimentul instrucțiunilor GRANT și REVOKE. Pentru a putea folosi comanda reload, trebuie să dispuneți de privilegiul RELOAD.

Funcții utilitare din modulul DBI

Aceste funcții sunt invocate sub forma `DBI::nume_functie()`, nu sub forma `DBI->nume_functie()`.

- `@bool = DBI::looks_like_number(@tablou);`

  Preia o listă cu valori și returnează un tablou care conține un membru pentru fiecare element al listei. Fiecare membru indică dacă argumentul corespunzător arată ca un număr: „adevărat" dacă această similitudine există, „fals" în caz contrar, undef dacă argumentul este nedefinit sau vid.

- `$str = DBI::neat($valoare [, $lungmax]);`

  Returnează un șir care conține o reprezentare estetic formatată a argumentului $valoare. Șirurile sunt încadrate între ghilimele, numerele nu. (Rețineți însă că numerele încadrate între ghilimele vor fi considerate șiruri.) Valorile nedefinite sunt raportate ca undef, iar caracterele imposibil de afișat sunt raportate sub forma _.

  Argumentul $lungmax controlează lungimea maximă a rezultatului. Dacă rezultatul este mai lung decât $lungmax, va fi redus la $lungmax – 4 caractere și i se vor adăuga puncte de suspensie (...). Dacă $lungmax este 0, undef sau lipsește, valoarea sa prestabilită va fi `$DBI::neat_maxlen(400)`.

  Nu folosiți metoda neat() pentru construcția interogărilor; dacă trebuie să executați încadrări între ghilimele sau modificări ale semnificației unor caractere speciale, folosiți în schimb metoda quote().

- `$str = DBI::neat_list(\@listref [, $lungmax [, $sep_camp]]);`

  Apelează metoda neat() pentru fiecare element din listă indicat de primul argument, îl unește cu șirul de separare $sep_camp și returnează rezultatul sub formă de șir unic.

  Argumentul $lungmax este aplicat argumentelor individuale, nu șirului care rezultă din apelurile la metoda neat().

  Dacă argumentul $sep_camp lipsește, valoarea sa prestabilită este ", ".

Atribute DBI

DBI furnizează informații despre atribute la numeroase nivele. Majoritatea atributelor sunt asociate cu variabile de manipulare a bazelor de date sau a instrucțiunilor, dar nu cu ambele. Unele atribute, precum PrintError și RaiseError, pot fi asociate cu variabile de manipulare a bazelor de date sau a instrucțiunilor. În general, fiecare variabilă de manipulare își are propriile sale atribute, dar unele atribute care conțin informații despre erori, precum err și errstr, au un caracter dinamic, în sensul că se asociază cu variabila de manipulare folosită cel mai recent.

Atribute generale ale variabilelor de manipulare

Aceste atribute por fi aplicate variabilelor de manipulare individuale sau pot fi specificate în parametrul %attr la metode care preiau un parametru pentru a controla modul de operare a metodei. Dacă aceste atribute sunt folosite cu metoda connect(), prelucrarea metodelor DBI este influențată global, pe toată întinderea scriptului.

- `$h->{'ChopBlanks'}`

  Determină dacă metodele de preluare a rândurilor vor amputa caracterele blanc finale ale valorilor din coloanele CHAR. Opțiunea ChopBlanks este dezactivată în mod prestabilit pentru majoritatea driverelor de baze de date, dar pentru MySQL starea acestei opțiuni nu are nici o importanță, deoarece serverul amputează întotdeauna caracterele blanc de la sfârșitul valorilor CHAR.

- `$h->{'PrintError'}`

  Dacă opțiunea este activată, apariția unei erori legate de DBI determină afișarea unui mesaj de avertizare. În mod prestabilit, opțiunea PrintError este activată.

- `$h->{'RaiseError'}`

  Dacă opțiunea este activată, apariția unei erori legate de DBI determină terminarea automată a execuției scriptului. În mod prestabilit, opțiunea PrintError este dezactivată.

Atribute dinamice

Aceste atribute sunt asociate variabilei de manipulare celei mai recent folosite, reprezentată prin expresia $h în descrierile următoare:

- `$DBI::err`

  Este o operație similară cu apelarea metodei $h->err().

- `$DBI::errstr`

  Este o operație similară cu apelarea metodei $h->errstr().

- `$DBI::rows`

  Este o operație similară cu apelarea metodei $h->rows().

Atribute ale variabilelor de manipulare a bazelor de date specifice sistemului MySQL

Aceste atribute sunt specifice driverului DBI pentru MySQL, DBD::mysql.

- `$str = $dbh->{'info'};`

  Acest atribut conține aceleași informații care sunt returnate de către funcția din interfața API C mysql_info(). Examinați descrierea acestei funcții în Anexa F.

- `$rv = $dbh->{'mysql_insertid'};`

  Valoarea AUTO_INCREMENT cel mai recent generată de conexiunea asociată cu $dbh.

- `$rv = $dbh->{'thread_id'};`

  Numărul de fir al conexiunii asociate cu $dbh.

Atribute ale variabilelor de manipulare a instrucțiunilor

În general, aceste atribute se aplică la o interogare SELECT (sau de tip SELECT) și nu sunt valide decât atunci când interogarea a fost transferată metodei prepare() pentru a obține o variabilă de manipulare a instrucțiunii și când a fost apelată metoda execute() pentru variabila respectivă. În plus, metoda finish() poate invalida unele atribute ale instrucțiunii.

Multe din aceste atribute au valori care constituie o referință la un tablou de valori, câte o valoare pentru fiecare coloană a interogării. Numărul de elemente al tabloului este dat de atributul $sth->{'NUM_OF_FIELDS'}. Pentru un atribut de instrucțiune atr_instr care este o referință la un tablou, puteți face referire la întregul tablou sub forma @{$sth->{atr_instr}} sau puteți parcurge ciclic elementele tabloului astfel:

```
for (my $i = 0; $i < $sth->{'NUM_OF_FIELDS'}; $i++)
{
    $value = $sth->{atr_instr}->[$i];
}
```

- $sth->{'NAME'};

  O referință la un tablou de șiruri care indică numele fiecărei coloane. Mărimea literelor folosite pentru nume (majuscule sau minuscule) este cea specificată în instrucțiunea SELECT.

- $sth->{'NAME_lc'};

  O referință la un tablou de șiruri care indică numele fiecărei coloane. Numele sunt returnate sub formă de șiruri scrise cu minuscule.

- $sth->{'NAME_uc'};

  O referință la un tablou de șiruri care indică numele fiecărei coloane. Numele sunt returnate sub formă de șiruri scrise cu majuscule.

- $sth->{'NULLABLE'};

  O referință la un tablou de șiruri care indică dacă fiecare coloană poate conține sau nu valoarea NULL. Valorile pentru fiecare element pot fi 0 (nu), 1 (da) sau 2 (necunoscut). Cel puțin așa se precizează în documentația DBI... Dacă în coloană nu pot exista valori NULL, se pare că valoarea returnată este de fapt un șir vid.

- $sth->{'NUM_OF_FIELDS'};

  Numărul de coloane care vor fi returnate de o instrucțiune pregătită, respectiv zero pentru o instrucțiune care nu este de tip SELECT.

- $sth->{'NUM_OF_PARAMS'};

  Numărul câmpurilor de înlocuire dintr-o instrucțiune pregătită.

- $sth->{'PRECISION'};

  O referință la un tablou de șiruri care indică precizia pentru fiecare coloană. DBI folosește termenul de „precizie" în sensul utilizat de ODBC, ceea ce pentru MySQL înseamnă lățimea maximă a coloanei. Pentru coloanele numerice, aceasta este lățimea de afișare. Pentru coloanele de tip șir, este lungimea maximă a coloanei.

- $sth->{'SCALE'};

  O referință la un tablou de șiruri care indică scara fiecărei coloane. DBI folosește termenul de „scară" în sensul utilizat de ODBC, ceea ce pentru MySQL înseamnă numărul cifrelor după virgulă, pentru coloanele cu virgulă mobilă. Pentru alte coloane, valoarea scării este 0.

- $sth->{'STATEMENT'};

  Textul instrucțiunii asociate cu $sth. Textul este cel văzut de metoda prepare() înainte de începerea operațiilor de substituire a câmpurilor de înlocuire.

- $sth->{'TYPE'};

  O referință la un tablou de valori care indică tipul numeric al fiecărei coloane.

Atribute ale variabilelor de manipulare a instrucțiunilor specifice sistemului MySQL

Majoritatea acestor atribute ale variabilelor de manipulare a instrucțiunilor trebuie să fie considerate ca fiind numai pentru citire, iar accesul la acestea trebuie să se realizeze după invocarea metodei execute(). Excepțiile sunt mysql_store_result și mysql_use_result. DBD::mysql furnizează posibilitatea de a controla stilul de prelucrare a setului de rezultate folosit de scriptul dumneavoastră. Atributele variabilelor de manipulare a instrucțiunilor mysql_store_result și mysql_use_result selectează maniera de prelucrare a setului de rezultate de către funcțiile din interfața API C mysql_store_result() și mysql_use_result(). Pentru o prezentare a acestor funcții și a diferențelor dintre ele, consultați Anexa F.

În mod prestabilit, DBI folosește mysql_store_result(), dar dumneavoastră puteți activa atributul mysql_use_result, care indică modulului DBI să folosească în schimb funcția mysql_use_result(). Următoarele operații trebuie executate după prepare(), dar înainte de execute():

```
$sth = $dbh->prepare (...);
$sth->{mysql_use_result} = 1;
$sth->execute();
```

Numeroase atribute specifice sistemului MySQL care erau disponibile în versiunile mai vechi ale driverului DBD::mysql sunt acum scoase din uz și au fost înlocuite de forme preferate mai recente, așa cum se poate vedea în tabelul G.2. Dacă versiunea dumneavoastră a driverului DBD::mysql este veche și nu acceptă atributele mai recente, încercați formele scoase din uz sau treceți la o versiune mai recentă.

Rețineți că insertid este un atribut al variabilelor de manipulare a instrucțiunilor, în timp ce forma sa mai recentă, mysql_insertid, este un atribut al variabilelor de manipulare a bazelor de date.

Tabelul G.2 Atribute specifice sistemului MySQL scoase din uz

Atribut scos din uz	Atribut preferat
insertid	mysql_insertid
is_blob	mysql_is_blob
is_key	mysql_is_key
is_not_null	mysql_is_not_null
is_num	mysql_is_num
is_pri_key	mysql_is_pri_key
length	PRECISION
max_length	mysql_max_length
table	mysql_table

- `$sth->{'mysql_is_blob'};`

  O referință la un tablou de valori care indică dacă fiecare coloană este sau nu de tip BLOB.
- `$sth->{'mysql_is_key'};`

  O referință la un tablou de valori care indică dacă fiecare coloană face sau nu parte dintr-o cheie non-unică.
- `$sth->{'mysql_is_not_null'};`

  O referință la un tablou de valori care indică dacă fiecare coloană poate conține sau nu valoarea NULL. O valoare „fals" arată că respectiva coloană poate conține valori NULL. Acest atribut conține informații care reprezintă „inversele" informațiilor conținute în atributul NULLABLE.
- `$sth->{'mysql_is_num'};`

  O referință la un tablou de valori care indică dacă fiecare coloană este sau nu de tip numeric.
- `$sth->{'mysql_is_pri_key'};`

  O referință la un tablou de valori care indică dacă fiecare coloană face sau nu parte dintr-o cheie primară (PRIMARY KEY).
- `$sth->{'mysql_max_length'};`

  O referință la un tablou de valori care indică lungimea maximă efectivă a valorilor coloanelor din setul de rezultate.
- `$sth->{'mysql_store_result'};`

  Dacă atributul `mysql_store_result` este activat (are valoarea 1), seturile de rezultate sunt regăsite din serverul MySQL care folosește funcția API C `mysql_store_result()`, nu funcția `mysql_use_result()`. Pentru o prezentare a acestor funcții și a diferențelor dintre ele, consultați Anexa F.

  Rețineți că utilizarea acestui atribut determină invalidarea unor atribute, între care și `mysql_max_length`. De asemenea, este invalidată utilizarea metodei `rows()`, deși este mai bine să numărați rândurile atunci când le preluați.

  În cazul în care activați atributul `mysql_store_result`, procedați astfel după ce ați invocat metoda `prepare()` și înainte de invocarea metodei `execute()`.
- `$sth->{'mysql_table'}`

  O referință la un tablou de valori care indică numele tabelului din care provine fiecare coloană. În cazul unei coloane cu valori calculate, numele tabelului constă dintr-un șir vid.
- `$sth->{'mysql_type'}`

  O referință la un tablou de valori care indică numărul tipului MySQL pentru fiecare coloană din setul de rezultate.
- `$sth->{'mysql_type_name'}`

  O referință la un tablou de valori care indică numele tipului MySQL pentru fiecare coloană din setul de rezultate.
- `$sth->{'mysql_use_result'}`

  Dacă variabila `mysql_use_result` este activată (primește valoarea 1), seturile de rezultate sunt regăsite din serverul MySQL care folosește funcția API C `mysql_use_result()`, nu funcția `mysql_store_result()`. Pentru o prezentare a acestor funcții și a diferențelor dintre ele, consultați Anexa F.

  Rețineți că utilizarea acestui atribut determină invalidarea unor atribute precum `mysql_max_length`. De asemenea, este invalidată utilizarea metodelor `rows()`, deși se recomandă numărarea rândurilor în momentul preluării acestora.

  În cazul în care activați atributul `mysql_use_result`, procedați astfel după ce ați invocat metoda `prepare()` și înainte de invocarea metodei `execute()`.

Variabile de mediu DBI

DBI consultă numeroase variabile de mediu, prezentate în tabelul G.3. Toate aceste variabile, cu excepția variabilei DBI_TRACE, sunt folosite de către metoda `connect()`. Variabila DBI_TRACE este folosită de către metoda `trace()`.

Tabelul G.3 Variabile de mediu DBI

Nume	Semnificație
DBI_DRIVER	Numele driverului de nivel DBD ("mysql" pentru MySQL)
DBI_DSN	Numele sursei de date
DBI_PASS	Parolă
DBI_TRACE	Nivel de urmărire și/sau nivelul fișierului cu datele de ieșire ale procesului de urmărire
DBI_USER	Numele utilizatorului

ANEXA H

Referință API PHP

Această anexă descrie interfața PHP de programare a aplicațiilor pentru MySQL. Interfața API constă dintr-un set de funcții pentru comunicarea cu serverele MySQL și pentru accesul la bazele de date.

Această anexă servește drept referință, deci include numai scurte fragmente de program care ilustrează utilizarea funcțiilor din interfața API pentru PHP. Pentru scripturile client complete și note privind modul de redactare a acestora, vezi capitolul 8, „Interfața API pentru PHP". Funcțiile descrise aici sunt cele care au o legătură directă cu MySQL. Manualul PHP (care în prezent are aproape 800 de pagini) conține mai mult de 600 de pagini cu materiale de referință, deci este evident că această anexă trebuie să se limiteze la a rezuma o mică parte din caracteristicile PHP. Pentru a vă procura manualul PHP complet, vizitați situl Web PHP de la următoarea locație: `http://www.php.net/`

Redactarea scripturilor PHP

Scripturile PHP sunt fișiere text simplu, care pot conține o combinație între programe HTML și PHP. Scriptul este interpretat astfel încât să producă o pagină Web care este trimisă la client. Codul HTML este copiat la ieșire fără a fi interpretat. Codul PHP este interpretat și înlocuit cu datele de ieșire pe care le produce programul (care pot chiar lipsi).

PHP începe prin a interpreta un fișier în modul HTML. Puteți comuta în interiorul și în afara modului cod HTML folosind etichete speciale, care semnifică începutul și sfârșitul codului PHP. În interiorul unui fișier, puteți trece de la un mod la altul de oricâte ori doriți. PHP înțelege patru tipuri de etichete, deși unele dintre ele trebuie să fie activate în mod explicit, dacă doriți să le folosiți. O modalitate de activare a etichetelor constă în punerea lor în funcțiune în fișierul de inițializare PHP, `php3.ini`. Amplasamentul acestui fișier este dependent de sistem; deseori, acest fișier se găsește în `/usr/local/lib`.

PHP înțelege următoarele stiluri de etichete:

- Stilul prestabilit folosește etichetele `<?php` și `?>`:

```
<?php echo ("Ceva programe PHP pe-aici"); ?>
```

- Stilul cu etichetă de deschidere scurtă folosește etichetele `?` și `?`:

```
<? echo ("Ceva programe PHP pe-aici"); ?>
```

  Acest stil de etichetă poate fi activat cu o directivă în fișierul de inițializare PHP:

```
short_open_tag = On;
```


- Stilul compatibil Active Server Page folosește etichetele `<%` și `%>`:

```
<% echo ("Ceva programe PHP pe-aici"); %>
```

  Etichetele în stil ASP pot fi activate cu o directivă în fișierul de inițializare PHP:

```
asp_tags = On;
```

  Suportul pentru etichetele ASP a fost introdus în versiunea PHP 3.0.4.

- Dacă folosiți un editor HTML care nu înțelege celelalte stiluri de etichete, puteți folosi etichetele `<SCRIPT>` și `</SCRIPT>`:

```
<SCRIPT LANGUAGE="php"> echo ("Ceva programe PHP pe-aici"); </SCRIPT>
```

Funcții

În continuare vom prezenta fiecare dintre funcțiile limbajului PHP legate de MySQL. Parametrii opționali sunt indicați prin paranteze drepte ([ ]).

Multe funcții preiau un parametru opțional `link_id` (identificator de legătură), care indică o conexiune cu un server MySQL. Dacă identificatorul de legătură lipsește dintr-un apel la o funcție, atunci se va folosi conexiunea cel mai recent deschisă. Dacă nu este specificată, respectiv nu este deschisă nici o conexiune, unele funcții vor încerca să stabilească o conexiune.

Un parametru `result_id` indică un identificator al setului de rezultate, returnat în mod caracteristic de funcțiile `mysql_db_query()` și `mysql_query()`.

Unele funcții generează un mesaj de eroare în cazul producerii unei erori, în afară de returnarea unei valori de stare. În contextele Web, acest mesaj apare în datele de ieșire trimise browserului client, ceea ce s-ar putea să nu intre în vederile dumneavoastră. Un nume de funcție poate fi precedat de caracterul `@` pentru a elimina mesajul de eroare (posibil criptic) pe care funcția l-ar genera în mod normal. De exemplu, pentru a elimina mesajul de eroare al unui apel la funcția `mysql_pconnect()` astfel încât să puteți raporta eșecul într-o manieră mai convenabilă, puteți proceda astfel:

```
<?php
$link = @mysql_pconnect ("pit-viper.snake.net","paul", "secret")
    or die ("Conectarea nu se poate realiza");
print ("Conectare reusita");
?>
```

O altă modalitate de a elimina mesajele de eroare este de a folosi funcția `error_reporting()`:

```
<?php
    error_reporting(0); # suprima toate mesajele de eroare
    $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
        or die ("Conectarea nu se poate realiza");
    print ("Conectare reusita");
?>
```

Multe dintre scripturile date ca exemple în această anexă afișează mesajul `"Conectare reusita"` după stabilirea unei conexiuni cu serverul MySQL. Motivul pentru care procedează astfel este acela de a se asigura că scriptul afișează date de ieșire oarecare, în caz că doriți să încercați scriptul. Un script PHP care nu produce date de ieșire determină afișarea de către unele browsere a unui mesaj de avertizare de tipul „pagina nu conține date".

În descrierile prezentate în continuare, termenul de „interogare SELECT" trebuie asimilat atât cu o interogare SELECT, cât și cu orice altă interogare care returnează rânduri, precum DESCRIBE, EXPLAIN sau SHOW.

Rutine de gestiune a conexiunii

Rutinele din această secțiune vă permit să deschideți și să închideți conexiunile cu serverul MySQL.

- int

mysql_close([int link_id]);

Închide conexiunea cu serverul MySQL identificat de link_id. Dacă nu este specificată nici o conexiune, mysql_close() închide conexiunea cel mai recent deschisă.

Funcția mysql_close() returnează „adevărat" în caz de reușită, respectiv „fals" la erori. Pentru conexiunile persistente deschise cu mysql_pconnect(), funcția mysql_close() ignoră cererea de închidere, dar returnează „adevărat". Dacă veți închide o conexiune, trebuie să o deschideți folosind mysql_connect() în locul funcției mysql_pconnect().

```
<?php
    $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
        or die ("Conectarea nu se poate realiza");
    print ("Conectare reusita");
    mysql_close($link);
?>
```

- int

mysql_connect([string nume_gazda] [, string nume_utilizator [, string parola]]);

Deschide o conexiune cu serverul MySQL pe gazda nume_gazda pentru utilizatorul nume_utilizator având parola dată. Returnează identificatorul de legătură (un număr pozitiv) asociat cu noua conexiune; returnează „fals" în cazul producerii unei erori.

Începând de la PHP 3.0.4, parametrul care desemnează numele gazdei poate fi specificat cu un număr de port opțional, în forma "nume_gazda:numar_port". Începând de la versiunea PHP 3.0.10, dacă numele gazdei este localhost, parametrul care indică numele gazdei poate fi specificat cu un nume de cale opțional pentru a preciza calea de acces a soclului de domeniu UNIX, sub forma "localhost:nume_soclu". Soclul trebuie specificat ca o cale de acces completă.

Dacă parametrul care indică numele gazdei lipsește, valoarea prestabilită a acestuia este localhost. Dacă parametrul cu numele utilizatorului lipsește sau este vid, valoarea prestabilită este numele de utilizator sub contul căruia rulează PHP. (Acesta este numele utilizatorului sub contul căruia rulează serverul Web, dacă PHP rulează ca modul Apache, respectiv numele utilizatorului sub contul căruia rulează PHP, dacă PHP rulează ca program autonom.) Dacă parametrul parolă lipsește sau este vid, atunci se va trimite o parolă vidă.

În timp ce conexiunea este deschisă, dacă funcția mysql_connect() este apelată cu aceiași parametri de conexiune (numele gazdei, numele de utilizator, parolă), nu se va genera o conexiune nouă; funcția mysql_connect() va returna în schimb identificatorul legăturii existente.

Conexiunea poate fi închisă prin apelarea funcției mysql_close(). În cazul în care conexiunea este deschisă atunci când scriptul își încheie execuția, conexiunea va fi închisă automat.

```
<?php
    $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
        or die ("Conectarea nu se poate realiza");
    print ("Conectare reusita");
    mysql_close($link);
?>
```

- int

mysql_pconnect([string nume_gazda] [, string nume_utilizator [, string parola]]);

Funcția mysql_pconnect() este asemănătoare funcției mysql_connect(), cu excepția faptului că deschide o conexiune persistentă. Cu alte cuvinte, conexiunea rămâne deschisă la terminarea execuției scriptului. Dacă se face un alt apel la funcția mysql_pconnect() cu aceiași parametri de conexiune (numele gazdei, numele de utilizator, parola) în timp ce conexiunea este deschisă, atunci conexiunea se va reutiliza. Astfel se evită suprasarcina de distrugere și redeschidere a conexiunilor, acest tip de conexiuni fiind mai eficient decât conexiunile non-persistente.

Conexiunile persistente au sens numai atunci când PHP se execută ca modul în cadrul unui server Web care continuă să ruleze și după ce PHP și-a încheiat execuția. Într-un script executat de o versiune autonomă a sistemului PHP, conexiunea este închisă la terminarea scriptului, deoarece tot atunci se termină și procesul PHP.

Apelarea funcției mysql_close() pe o conexiune persistentă este o operație lipsită de sens; în acest caz, funcția mysql_close() returnează „adevărat", dar lasă conexiunea deschisă.

```
<?php
    $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
        or die ("Conectarea nu se poate realiza");
    print ("Conectare reusita");
?>
```

Rutine de raportare a stării și a erorilor

Funcțiile mysql_errno() și mysql_error() returnează numerele erorilor și mesaje de eroare pentru funcțiile PHP legate de MySQL. Cu toate acestea, nici una din funcții nu poate furniza informații privind erorile în absența unui identificator de legătură corect. Aceasta înseamnă că aceste funcții nu se pot utiliza pentru raportarea rezultatului unor apeluri ratate la funcțiile mysql_connect() sau mysql_pconnect(), deoarece nu există

nici un identificator de legătură decât atunci când s-a reușit stabilirea unei legături. Dacă doriți să obțineți mesajul de eroare MySQL pentru tentativele de conexiune eșuate, activați variabila `track_errors` cu o directivă în fișierul de inițializare PHP:

```
track_errors = On;
```

Apoi reporniți serverul Web, dacă rulați PHP ca modul Apache. După ce ați procedat astfel, puteți obține șirul de eroare pentru încercările de conexiune eșuate prin referirea la variabila `$php_errormsg`:

```
<?php
    $link = @mysql_connect ("gazda_rea", "utilizator_rau", "parola_rea")
            or die ("Conectare imposibila: " . $php_errormsg);
    print ("Conectare reusita");
?>
```

- int

mysql_errno([int link_id]);

Pentru conexiunea dată, returnează numărul erorii pentru funcția conexă sistemului MySQL care a returnat cel mai recent un mesaj de eroare. O valoare egală cu zero indică faptul că nu s-a produs nici o eroare.

```
<?php
    $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
        or die ("Conectarea nu se poate realiza");
    print ("Conectare reusita");
    $query = "SELECT * FROM presedinte";
    $result = mysql_query($query)
        or die ("interogare ratata, cod de eroare = " . mysql_errno());
?>
```

- string

mysql_error([int link_id]);

Pentru conexiunea dată, returnează un șir care conține mesajul de eroare pentru funcția conexă sistemului MySQL care a returnat cel mai recent un mesaj de eroare. O valoare egală cu zero indică faptul că nu s-a produs nici o eroare.

```
<?php
    $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
        or die ("Conectarea nu se poate realiza");
    print ("Conectare reusita");
    $query = "SELECT * FROM presedinte";
    $result = mysql_query($query)
        or die ("interogare ratata, mesaj de eroare = " .
            mysql_error());
?>
```


Rutine de construcție și execuție a interogărilor

Rutinele din această secțiune sunt folosite pentru a emite interogări spre serverul MySQL.

- int

mysql_db_query(string nume_db, string interogare [, int link_id]);

Funcția `mysql_db_query()` este similară cu funcția `mysql_query()`, cu deosebirea că mai preia un argument suplimentar care indică numele unei baze de date, pe care o transformă în bază de date prestabilită înainte de a executa interogarea. (Comparați exemplul următor cu exemplul furnizat pentru funcția `mysql_query()`.)

```
<?php
    $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
        or die ("Conectarea nu se poate realiza");
    print ("Conectare reusita");
    $query = "SELECT * FROM presedinte";
    $result = mysql_db_query("samp_db", $query)
        or die ("Interogare ratata");
?>
```

- int

mysql_list_dbs([int link_id]);

Returnează un identificator de rezultat pentru un set de rezultate care constă din numele bazelor de date cunoscute de server pe conexiunea dată, câte un nume de bază de date în fiecare rând al setului de rezultate. Returnează „fals" dacă se produce o eroare. O bază de date prestabilită nu trebuie selectată. Setul de rezultate poate fi prelucrat de oricare dintre funcțiile obișnuite de preluare a rândurilor sau de către funcția `mysql_tablename()`.

```
<?php
    $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
        or die ("Conectarea nu se poate realiza");
    $result = mysql_list_dbs()
        or die ("Interogare ratata");
    print ("Baze de date (folosind mysql_fetch_row()):<BR>\n";
    while ($row = mysql_fetch_row($result))
        printf("%s<BR>\n", $row[0]);
    $result = mysql_list_dbs()
        or die ("Interogare ratata");
    print ("Baze de date (folosind mysql_tablename()):<BR>\n";
    for ($i = 0; $i < mysql_num_rows($result); $i++)
        printf("%s<BR>\n", mysql_tablename($result, $i));
?>
```

- int

mysql_list_fields(string nume_db, string nume_tabel [, int link_id]);

Returnează un identificator de rezultat pentru un set de rezultate care conține informații despre coloanele dintr-un tabel; returnează „fals“ dacă se produce o eroare. O bază de date prestabilită nu trebuie să fie selectată. Parametrii `nume_db` și `nume_tabel` identifică baza de date și tabelul care vă interesează. Identificatorul de rezultat se poate folosi cu oricare din funcțiile `mysql_field_flags()`, `mysql_field_len()`, `mysql_field_name()` și `mysql_field_type()`.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        $result = mysql_list_fields("samp_db", "membru")
                or die ("Interogare ratata");
        print ("Informatii despre coloanele tabelului membru:BR\n";
        for ($i = 0; $i < mysql_num_fields($result);$i++)
        {
           printf("coloana %d:", $i);
           printf(" nume %s,\n", mysql_field_name($result, $i));
           printf(" lungime %d,\n", mysql_field_len($result, $i));
           printf(" tip %s,\n", mysql_field_type($result, $i));
           printf(" flags %s, \n", mysql_field_flags($result, $i));
           print("<BR>\n");
        }
?>
```

- int

mysql_list_tables(string nume_db [, int link_id]);

Returnează un identificator de rezultat pentru un set de rezultate care constă din numele tabelelor din baza de date specificată, câte un nume de tabel în fiecare rând al setului de rezultate. Returnează „fals“ dacă se produce o eroare. O bază de date prestabilită nu trebuie selectată. Setul de rezultate poate fi prelucrat de oricare dintre funcțiile obișnuite de preluare a rândurilor sau de către funcția `mysql_tablename()`.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        $result = mysql_list_tables("samp_db")
                or die ("Interogare ratata");
        print ("tabelele din samp_db (cu mysql_fetch_row()):<BR>\n";
        while ($row = mysql_fetch_row($result))
                printf("%s<BR>\n", $row[0]);
        $result = mysql_list_tables("samp_db")
                or die ("Interogare ratata");
        print ("tabelele din samp_db (cu mysql_tablename()):<BR>\n";
        for ($i = 0; $i < mysql_num_rows($result); $i++)
                printf("%s<BR>\n", mysql_tablename($result, $i));
?>
```


- int

mysql_query(string interogare [, int link_id]);

Trimite șirul de interogare către serverul MySQL pe conexiunea dată. Pentru instrucțiunile `DELETE`, `INSERT`, `REPLACE` și `UPDATE`, funcția `mysql_query()` returnează „adevărat“ pentru succes, respectiv „fals“ dacă se produce o eroare. Pentru o interogare reușită, puteți apela funcția `mysql_affected_rows()` pentru a descoperi numărul rândurilor modificate.

Pentru instrucțiunile `SELECT`, funcția `mysql_query()` returnează un identificator sigur al setului de rezultate în caz de reușită, respectiv „fals“ dacă se produce vreo eroare. Pentru o interogare reușită, identificatorul de rezultat poate fi folosit cu diferitele funcții de prelucrare a setului de rezultate care preiau un argument `result_id`. Identificatorul poate fi transferat funcției `mysql_free_result()` pentru eliberarea tuturor resurselor asociate setului de rezultate.

O interogare „reușită“ se execută fără erori, dar reușita nu implică nimic legat de returnarea sau nereturnarea unor rânduri. Interogarea următoare este absolut corectă, dar nu returnează nici un rând:

```
SELECT * FROM presedinte WHERE 1 = 0
```

O interogare poate eșua din numeroase motive. De exemplu, poate fi eronată sintactic, incorectă din punct de vedere semantic sau ilegală, deoarece nu aveți permisiunea de a obține acces la tabelele specificate în interogare.

Dacă nu este specificat nici un identificator de legătură, se va folosi conexiunea cel mai recent deschisă. Dacă nu există nici o conexiune curentă, funcția `mysql_query()` încearcă să deschidă o conexiune ca și cum funcția `mysql_connect()` ar fi fost apelată fără argumente. Funcția `mysql_query()` eșuează dacă încercarea de conectare nu reușește.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        print ("Conectare reusita");
        mysql_select_db("samp_db")
                or die ("Nu poate selecta baza de date");
        $query = "SELECT * FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata");
?>
```

Rutine de prelucrare a setului de rezultate

Rutinele din această secțiune sunt folosite pentru regăsirea rezultatelor interogărilor. De asemenea, furnizează acces la informații despre rezultat, precum numărul rândurilor afectate sau metadatele aferente coloanelor setului de rezultate.

- int

mysql_affected_rows([int link_id]);

Returnează numărul rândurilor afectate (modificate) de către cea mai recentă instrucțiune DELETE, INSERT, REPLACE sau UPDATE executată pe conexiunea dată. Funcția mysql_affected_rows() returnează 0 dacă nu au fost modificate rânduri, respectiv –1 dacă s-a produs o eroare.

După o interogare SELECT, funcția mysql_affected_rows() returnează numărul rândurilor selectate. Totuși, la instrucțiunile SELECT se folosește de obicei funcția mysql_num_rows().

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
                or die ("Nu poate selecta baza de date");
        $query = "INSERT INTO membru (nume,prenume,data_expirare)"
                . " VALUES('Vasile','Cornel','2002-6-3')";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        printf ("%d randuri inserate\n",
                        mysql_affected_rows(),
                        mysql_affected_rows() = = 1 ? "" : "s");
?>
```

- int

mysql_data_seek(int result_id, int row_num);

Fiecare set de rezultate returnate de o interogare SELECT are un cursor de rând, care indică rândul ce trebuie returnat de următorul apel la funcțiile de preluare a rândurilor, în speță mysql_fetch_array(), mysql_fetch_object() sau mysql_fetch_row(). Funcția mysql_data_seek() fixează cursorul pentru setul de rezultate dat pe rândul dat. Numărul rândului trebuie să fie cuprins între 0 și mysql_num_rows()–1. Funcția mysql_data_seek() returnează „adevărat" dacă numărul rândului este corect, respectiv „fals" în caz contrar.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
                or die ("Nu poate selecta baza de date");
        $query = "SELECT nume,prenume FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        # preia randurile in ordine inversa
        for ($i = mysql_num_rows($result) - 1; $i >= 0; $i-)
        {
                if (!mysql_data_seek($result, $i))
                {
                        printf("Nu poate cauta in randul %d\n", $i);
                        continue;
                }
                if(!($row = mysql_fetch_object($result)))
                        continue;
                printf("%s %s<BR>\n", $row->nume, $row->prenume);
        }
        mysql_free_result($result);
?>
```


- array

mysql_fetch_array(int result_id, [, int result_type]);

Returnează următorul rând al setului de rezultate date sub forma unui tablou. Returnează „fals" dacă nu mai există rânduri. Tabloul conține valori stocate atât sub formă de indici numerici de coloană, cât și sub formă de chei asociative determinate de numele coloanelor. Cu alte cuvinte, fiecare valoare dintr-o coloană este accesibilă atât prin indexul său numeric, cât și prin numele său. Indicii asociativi sunt sensibili la diferența între majuscule și minuscule și trebuie scriși cu aceeași mărime de literă folosită pentru desemnarea coloanelor în cadrul interogării. Să presupunem că emiteți următoarea interogare:

```
SELECT nume,prenume FROM presedinte
```

Dacă preluați rândurile dintr-un set de rezultate într-un tablou denumit $row, elementele tabloului sunt accesibile după cum urmează:

```
$row[0]              Contine valoarea nume
$row[1]              Contine valoarea prenume
$row["nume"]         Contine valoarea nume
$row["prenume"]      Contine valoarea prenume
```

Cheile nu conțin numele tabelelor pentru coloanele corespunzătoare, deci, dacă selectați coloane cu același nume din tabele diferite, veți obține un conflict de nume. Prioritatea este acordată coloanei specificate ultima în lista de coloane specificată de interogare. Pentru a obține accesul la coloana ascunsă, folosiți indexul numeric al acesteia sau scrieți interogarea pentru a furniza coloanei un alias.

Parametrul result_type poate fi MYSQL_ASSOC (returnează valorile numai în funcție de indicii de nume), MYSQL_NUM (returnează valorile numai în funcție de indicii numerici), respectiv MYSQL_BOTH (returnează valorile în funcție de ambele categorii de indici). Dacă result_type nu este specificat, valoarea prestabilită este MYSQL_BOTH.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
                or die ("Nu poate selecta baza de date");
        $query = "SELECT nume,prenume FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata");
while ($row = mysql_fetch_array($result))
```

Continuare

Continuare

```
        {
                # afiseaza fiecare nume de doua ori, o data folosind
                # indici numerici, iar alta data folosind indici
                # asociativi (de nume)
                printf("%s %s<BR>\n", $row[0], $row[1]);
                printf("%s %s<BR>\n", $row["nume"], $row["prenume"]);
        }
        mysql_free_result();
        ?>
```

- object

mysql_fetch_field(int result_id [, int col_num]);

Returnează metadatele referitoare la coloana dată din setul de rezultate; returnează „fals“ dacă nu există o asemenea coloană. Dacă parametrul `col_num` este omis, apelurile succesive la funcția `mysql_fetch_field()` returnează informații despre coloanele succesive ale setului de rezultate. Valoarea returnată este „fals“ dacă nu au mai rămas coloane. Dacă parametrul `col_num` este specificat, valoarea acestuia trebuie să fie cuprinsă între 0 și `mysql_num_fields()-1`. În acest caz, `mysql_fetch_field()` returnează informații despre coloana dată, respectiv „fals“ dacă valoarea parametrului `col_num` nu se află în domeniul specificat.

Informațiile sunt returnate sub forma unui obiect care are proprietățile prezentate în tabelul H.1.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
                or die ("Nu poate selecta baza de date");
        $query = "SELECT * FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        # preia metadatele coloanei
        for ($i = 0; $i < mysql_num_fields($result); $i++)
        {
                printf("Informatii pentru coloana %d:<BR>\n", $i);
                $meta = mysql_fetch_field($result);
                if(!$meta)
                {
                        print("Nu exista informatii<BR>\n");
                        continue;
                }
                print("<PRE>\n");
                printf("blob:           %s/n", $meta->blob);
                printf("max_length:     %s/n", $meta->max_length);
                printf("multiple_key:   %s/n", $meta->multiple_key);
                printf("name:           %s/n", $meta->name);
                printf("not_null:       %s/n", $meta->not_null);
                printf("numeric:        %s/n", $meta->numeric);
                printf("primary_key:    %s/n", $meta->primary_key);
                printf("table:          %s/n", $meta->table);
                printf("type:           %s/n", $meta->type);
                printf("unique_key:     %s/n", $meta->unique_key);
                printf("unsigned:       %s/n", $meta->unsigned);
                printf("zerofill:       %s/n", $meta->zerofill);
                print("</PRE>\n");
        }
?>
```


Tabelul H.1 Proprietățile obiectului `mysql_fetch_field()`

Proprietate	Semnificație
blob	1, în cazul în care coloana este de tip BLOB, 0 în caz contrar
max_length	Lungimea celei mai mari valori din coloanele setului de rezultate
multiple_key	1, în cazul în care coloana face parte dintr-un index non-unic, 0 în caz contrar
name	Numele coloanei
not_null	1 în cazul în care coloana nu poate conține valori NULL, 0 în caz contrar
numeric	1 în cazul în care coloana este de tip numeric, 0 în caz contrar
primary_key	1 în cazul în care coloana face parte dintr-o cheie primară, 0 în caz contrar
table	Numele tabelului care conține coloana (are valoare vidă, în cazul în care coloana conține valori calculate)
type	Numele tipului coloanei
unique_key	1 în cazul în care coloana face parte dintr-un index UNIQUE, 0 în caz contrar
unsigned	1 în cazul în care coloana are atributul UNSIGNED, 0 în caz contrar
zerofill	1 în cazul în care coloana are atributul ZEROFILL, 0 în caz contrar

- array

mysql_fetch_lengths(int result_id);

Returnează un tablou care conține lungimile valorilor coloanelor din rândul cel mai recent preluat de oricare dintre funcțiile `mysql_fetch_array()`, `mysql_fetch_object()` sau `mysql_fetch_row()`. Returnează „fals“ dacă nu a fost încă preluat nici un rând, respectiv dacă s-a produs o eroare.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
```

Continuare

Continuare

```
                or die ("Nu poate selecta baza de date");
        $query = "SELECT * FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata");
    $row_num = 0;
    while (mysql_fetch_row(result))
    {
        ++$row_num;
        # obtine lungimile valorilor din coloane
        printf("Lungimile valorilor din randul %d:<BR>\n", $row_num);
        $len = mysql_fetch_lengths($result);
        if(!$len)
        {
                print("Nu exista informatii<BR>\n");
                break;
        }
        print("<PRE>\n");
        for ($i = 0; $i < mysql_num_fields($result); $i++)
                printf("coloana %d: %s\n", $i, $len[$i]);
        print("</PRE>\n");
    }
    ?>
```

- `object`

`mysql_fetch_object``(int result_id, [, int result_type]);`

Returnează rândul următor al setului de rezultate dat sub formă de obiect; returnează „fals" dacă nu au mai rămas rânduri. Valorile coloanelor sunt accesibile sub formă de proprietăți ale obiectului. Numele proprietăților sunt numele coloanelor selectate în interogarea care a generat setul de rezultate.

Parametrul `result_type` poate fi `MYSQL_ASSOC` (returnează valorile numai în funcție de indicii de nume), `MYSQL_NUM` (returnează valorile numai în funcție de indicii numerici), respectiv `MYSQL_BOTH` (returnează valorile în funcție de ambele categorii de indici). Dacă `result_type` nu este specificat, valoarea prestabilită este `MYSQL_BOTH`. (Mie unuia nu-mi este clar care este rostul specificării opțiunii `MYSQL_NUM`, din moment ce numerele nu pot constitui nume acceptate de proprietăți.)

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
                or die ("Nu poate selecta baza de date");
        $query = "SELECT nume,prenum FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        while ($row = mysql_fetch_object($result))
                printf("%s %s<BR>\n", $row->nume, $row->prenume);
        mysql_free_result($result);
?>
```


- `array`

`mysql_fetch_row``(int result_id);`

Returnează rândul următor al setului de rezultate sub formă de tablou; returnează „fals" dacă nu mai există rânduri.

Valorile din coloane sunt accesibile ca elemente ale tabloului, folosind indici de coloană cu valori cuprinse între 0 și `mysql_num_fields()-1`.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
                or die ("Nu poate selecta baza de date");
        $query = "SELECT nume,prenume FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        while ($row = mysql_fetch_row($result))
                printf("%s %s<BR>\n", $row[0], $row[1]);
        mysql_free_result($result);
?>
```

- `string`

`mysql_field_name``(int result_id, int col_num);`

Returnează numele coloanei date din setul de rezultate. Valoarea parametrului `col_num` trebuie să fie cuprinsă între 0 și `mysql_num_fields()-1`.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
                or die ("Nu poate selecta baza de date");
        $query = "SELECT * FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        # obtine numele coloanelor
        for ($i = 0; $i < mysql_num_fields($result); $i++)
        {
                printf("Numele coloanei %d: ", $i);
                $name = mysql_field_name($result, $i);
                if(!$name)
```

Continuare

Continuare

```
                print("Nu exista nici un nume<BR>\n");
            else
                print("$name<BR>\n");
        }
    ?>
```

- int

mysql_field_seek(int result_id, int col_num);

Stabilește indexul pentru apelurile ulterioare la funcția `mysql_fetch_field()`. Următorul apel către funcția `mysql_fetch_field()` care este emis fără un număr de coloană explicit va returna informații pentru coloana `col_num`. Returnează „adevărat" în cazul în care căutarea reușește și „fals" în caz contrar. Valoarea parametrului `col_num` trebuie să fie cuprinsă între 0 și `mysql_num_fields()-1`.

```
    <?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
            or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
            or die ("Nu poate selecta baza de date");
        $query = "SELECT * FROM presedinte";
        $result = mysql_query($query)
            or die ("Interogare ratata");
        # preia metadatele coloanei
        for ($i = 0; $i < mysql_num_fields($result); $i++)
        {
            printf("Informatiile pentru coloana %d:<BR>\n", $i);
            if (!mysql_field_seek($result, $i))
            {
                print("Nu poate cauta in coloana<BR>\n");
                continue;
            }
            $meta = mysql_fetch_field($result, $i);
            if(!$meta)
            {
                print("Nu exista informatii<BR>\n");
                continue;
            }
            print("<PRE>\n");
            printf("blob:           %s/n", $meta->blob);
            printf("max_length:     %s/n", $meta->max_length);
            printf("multiple_key:   %s/n", $meta->multiple_key);
            printf("name:           %s/n", $meta->name);
            printf("not_null:       %s/n", $meta->not_null);
            printf("numeric:        %s/n", $meta->numeric);
```



```
            printf("primary_key:    %s/n", $meta->primary_key);
            printf("table:          %s/n", $meta->table);
            printf("type:           %s/n", $meta->type);
            printf("unique_key:     %s/n", $meta->unique_key);
            printf("unsigned:       %s/n", $meta->unsigned);
            printf("zerofill:       %s/n", $meta->zerofill);
            print("</PRE>\n");
        }
    ?>
```

- string

mysql_field_table(int result_id, int col_num);

Returnează numele tabelului care conține coloana dată a setului de rezultate. Pentru coloanele care conțin valori calculate, numele este vid.

Valoarea parametrului `col_num` trebuie să fie cuprinsă între 0 și `mysql_num_fields()-1`.

```
    <?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
            or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
            or die ("Nu poate selecta baza de date");
        $query = "SELECT * FROM presedinte";
        $result = mysql_query($query)
            or die ("Interogare ratata");
        for ($i = 0; $i < mysql_num_fields($result); $i++)
        {
            printf("coloana %d:", $i);
            printf(" nume %s,\n", mysql_field_name($result, $i));
            printf(" tabel %s,\n", mysql_field_table($result, $i));
            print "<BR>\n";
        }
    ?>
```

- string

mysql_field_type(int result_id, int col_num);

Returnează numele tipului coloanei date a setului de rezultate. Numele tipurilor de date sunt enumerate în anexa B, „Referință de tipuri de coloane".

Valoarea parametrului `col_num` trebuie să fie cuprinsă între 0 și `mysql_num_fields()-1`.

```
    <?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
            or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
            or die ("Nu poate selecta baza de date");
        $query = "SELECT * FROM presedinte";
```

Continuare

Continuare

```
        $result = mysql_query($query)
                or die ("Interogare ratata");
        for ($i = 0; $i < mysql_num_fields($result); $i++)
        {
                printf("coloana %d:", $i);
                printf(" nume %s,\n", mysql_field_name($result, $i));
                printf(" tip %s,\n", mysql_field_type($result, $i));
                print "<BR>\n";
        }
?>
```

- string

mysql_field_flags(int result_id, int col_num);

Returnează metadatele aferente coloanei date din setul de rezultate, sub formă de şir; returnează „fals“ dacă se produce o eroare. Şirul constă din cuvinte separate prin spaţii, care indică dacă valorile indicator ale coloanei sunt „adevărat“. Pentru indicatoarele cu valoarea „fals“, cuvântul corespunzător nu se găseşte în şir. Tabelul H.2 enumeră cuvintele care se pot găsi în şir.

Valoarea parametrului col_num trebuie să fie cuprinsă între 0 şi mysql_num_fields()-1.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db");
        $query = "SELECT * FROM membru";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        for ($i = 0; $i < mysql_num_fields($result); $i++)
        {
                printf("coloana %d:", $i);
                printf(" nume %s,\n", mysql_field_name($result, $i));
                printf(" flags %s,\n", mysql_field_flags($result, $i));
                print "<BR>\n";
        }
?>
```

Tabelul H.2 Valorile returnate de funcţia mysql_field_flags()

Proprietate	Semnificaţie
auto_increment	Coloana are atributul AUTO_INCREMENT
binary	Coloana are atributul BINARY
blob	Coloana este de tip BLOB
enum	Coloana este de tip ENUM
multiple_key	Coloana face parte dintr-un index non-unic
not_null	Coloana nu poate conţine valori NULL
primary_key	Coloana face parte dintr-o cheie primară
timestamp	Coloana este de tip TIMESTAMP
unique_key	Coloana face parte dintr-un index UNIQUE
unsigned	Coloana are atributul UNSIGNED
zerofill	Coloana are atributul ZEROFILL

- int

mysql_field_len(int result_id, int col_num);

Returnează lungimea maximă posibilă a valorilor din coloana dată a setului de rezultate.

Valoarea parametrului col_num trebuie să fie cuprinsă între 0 şi mysql_num_fields()-1.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db");
        $query = "SELECT * FROM membru";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        for ($i = 0; $i < mysql_num_fields($result); $i++)
        {
                printf("coloana %d:", $i);
                printf(" nume %s,\n", mysql_field_name($result, $i));
                printf(" lungime %d,\n", mysql_field_len($result, $i));
                print "<BR>\n";
        }
?>
```

- int

mysql_free_result(int result_id);

Eliberează toate resursele asociate setului de rezultate dat. Seturile de rezultate sunt eliberate automat la terminarea execuţiei unui script, dar puteţi dori să apelaţi această funcţie în mod explicit, într-un script care generează numeroase seturi de rezultate. De exemplu, acest script va folosi o cantitate considerabilă de memorie:

```
<?php
        $link = mysql_connect("localhost","paul","secret");
        mysql_select_db("samp_db");
        for ($i = 0; $i < 10000; $i++)
        {
                $result = mysql_query("SELECT * FROM presedinte");
        }
?>
```

Dacă se adaugă un apel la funcția mysql_free_result() după apelul la funcția mysql_query(), se va reduce cantitatea de memorie utilizată pentru stocarea setului de rezultate până aproape de zero:

```
<?php
        $link = mysql_connect("localhost","paul","secret");
        mysql_select_db("samp_db");
        for ($i = 0; $i < 10000; $i++)
        {
                $result = mysql_query("SELECT * FROM presedinte");
                mysql_free_result($result);
        }
?>
```

- int

mysql_insert_id([int link_id]);

Returnează valoarea AUTO_INCREMENT generată de interogarea cel mai recent executată pe conexiunea dată. Returnează zero dacă pe durata vieții conexiunii respective nu a fost generată nici o asemenea valoare. În general, trebuie să apelați funcția mysql_insert_id() imediat după o interogare care se presupune că va genera o valoare nouă; dacă orice altă interogare se interpune între interogarea respectivă și locul unde doriți să folosiți valoarea în cauză, valoarea funcției mysql_insert_id() poate fi reinițializată la zero de către interogarea intercalată.

Rețineți că modul de comportare a funcției mysql_insert_id() diferă de acela al funcției SQL LAST_INSERT_ID(). Funcția mysql_insert_id() este menținută la client și valoarea sa este stabilită pentru fiecare interogare în parte. Valoarea funcției LAST_INSERT_ID()este menținută în server și se păstrează de la o interogare la alta.

Valorile returnate de funcția mysql_insert_id() sunt specifice conexiunii și nu sunt afectate de activitatea AUTO_INCREMENT desfășurată pe alte conexiuni.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
                or die ("Nu poate selecta baza de date");
        $query = "INSERT INTO membru (nume,prenume,data_expirare)"
                . " VALUES('Vasile','Cornel','2002-6-3')";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        printf("numarul noului membru: %d\n", mysql_insert_id());
?>
```

- int

mysql_num_fields(int result_id);

Returnează numărul de coloane din setul de rezultate dat.


```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Nu poate selecta baza de date");
        mysql_select_db("samp_db");
        $query = "SELECT * FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        printf("Numarul de coloane: %d\n", mysql_num_fields($result));
?>
```

- int

mysql_num_rows(int result_id);

Returnează numărul de rânduri din setul de rezultate dat.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Nu poate selecta baza de date");
        mysql_select_db("samp_db");
        $query = "SELECT * FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata");
        printf("Numarul de randuri: %d\n", mysql_num_rows($result));
?>
```

- int

mysql_result(int result_id, int row, mixed field);

Returnează o valoare din rândul dat al unui set de rezultate. Coloana este identificată prin parametrul field, care poate fi un index numeric de coloană sau numele coloanei specificat în interogare.

Această funcție este lentă; se recomandă utilizarea uneia din funcțiile mysql_fetch_array(), mysql_fetch_object() sau mysql_fetch_row().

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        mysql_select_db("samp_db")
                or die ("Nu poate selecta baza de date");
        $query = "SELECT nume,prenume FROM presedinte";
        $result = mysql_query($query)
                or die ("Interogare ratata);
        for ($i = 0; $i < mysql_num_rows($result); $i++)
        {
                for ($j = 0; $j < mysql_num_rows($result); $j++)
                {
                        if($j < 0)
                                print(" ");
```

Continuare

Continuare

```
                    print(mysql_result($result, $i, $j));
                }
                print "<BR>\n";
            }
            mysql_free_result($result);
    ?>
```

- string

mysql_tablename(int result_id, int row_num);

Dacă este dat un identificator de rezultat returnat de `mysql_list_dbs()`sau `mysql_list_tables()`, precum și un index de rând `row_num`, funcția returnează numele stocat în rândul dat al setului de rezultate.

Valoarea indexului de rând trebuie să fie cuprinsă între 0 și `mysql_num_rows()-1`.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        $result = mysql_list_tables("samp_db")
                or die ("Interogare ratata");
        print("tabelele din baza de date samp_db:<BR>\n");
        for ($i = 0; $i < mysql_num_rows($result); $i++)
                printf("%s<BR>\n", mysql_tablename($result, $i));
?>
```

Rutine pentru baze de date

Rutinele din această secțiune vă permit să creați și să distrugeți bazele de date. De asemenea, vă permit să selectați o bază de date prestabilită pentru sesiunea server curentă.

- int

mysql_create_db(string nume_db, [, int link_id]);

Indică serverului MySQL identificat prin `link_id` să creeze baza de date cu numele dat. Returnează „adevărat" dacă baza de date a fost creată cu succes, respectiv „fals" dacă s-a produs o eroare. Pentru a crea baza de date, trebuie să dispuneți de privilegiul `CREATE` aferent acesteia.

Este preferabil să folosiți funcția `mysql_query()` pentru a emite o instrucțiune `CREATE DATABASE`, decât să folosiți funcția `mysql_create_db()`.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        if (mysql_create_db("my_db"))
                print("Baza de date a fost creata cu succes\n");
        else
                printf("Eroare la crearea bazei de date: %s\n",
                        mysql_error());
?>
```


- int

mysql_drop_db(string nume_db, [, int link_id]);

Indică serverului MySQL identificat prin `link_id` să șteargă baza de date cu numele dat. Returnează „adevărat" dacă baza de date a fost eliminată cu succes, respectiv „fals" dacă s-a produs o eroare. Pentru a elimina baza de date, trebuie să dispuneți de privilegiul `DROP` aferent acesteia.

Fiți atent cu această funcție; dacă ștergeți o bază de date, ștearsă rămâne. Nu o mai puteți recupera.

Este preferabil să folosiți funcția `mysql_query()` pentru a emite o instrucțiune `DROP DATABASE`, decât să folosiți funcția `mysql_drop_db()`.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        if (mysql_drop_db("my_db"))
                print("Baza de date a fost eliminata cu succes\n");
        else
                printf("Eroare la eliminarea bazei de date: %s\n",
                        mysql_error());
?>
```

- int

mysql_select_db(string nume_db, [, int link_id]);

Selectează baza de date specificată pentru a o transforma în bază de date prestabilită pentru conexiunea dată. Returnează „adevărat" în caz de succes, respectiv „fals" dacă se produce o eroare.

Dacă nu este specificată nici o conexiune, se va folosi conexiunea curentă. Dacă nu există o conexiune curentă, funcția `mysql_select_db()` încearcă să deschidă o conexiune ca și cum funcția `mysql_connect()` ar fi fost apelată fără argumente. Funcția `mysql_select_db()` eșuează dacă încercarea de conexiune nu reușește.

```
<?php
        $link = @mysql_pconnect ("pit-viper.snake.net", "paul", "secret")
                or die ("Conectarea nu se poate realiza");
        print ("Conectarea s-a realizat cu succes");
        mysql_select_db("samp_db")
                or die ("Nu s-a putut selecta baza de date");
?>
```

Anexa I

Instrumente utile produse de terțe părți

Această anexă vă oferă o succintă vedere de ansamblu ilustrativă a tipurilor de instrumente create de terțe părți disponibile care vă pot ajuta să folosiți sistemul MySQL mai simplu și mai eficient. O listă cu programe mai cuprinzătoare și frecvent actualizată este disponibilă în situl Web al sistemului MySQL. Intenția anexei de față este pur și simplu de a vă pune la curent cu unele dintre tipurile de instrumente disponibile și de a vă încuraja să examinați personal câteva dintre acestea. Pentru mai multe informații, vizitați adresa `http://www.mysql.com/Contrib/`. Veți descoperi numeroase pachete care vă pot fi de ajutor. Aceste pachete sunt grupate în mai multe categorii:

- **Interfețe de programare a aplicațiilor în diverse limbaje și instrumente de programare.** MySQL este bine reprezentat de către interfețele de programare. În afară de interfețele Perl și PHP care au fost discutate în capitolele 7, „Interfața API pentru Perl DBI" și 8, „Interfața API pentru PHP", sunt disponibile interfețe pentru numeroase alte limbaje. Acestea includ C++, Delphi, Eiffel, Guile, Java, Matlab, Pike, Python, Ruby, Smalltalk și Tcl. Există și biblioteci container, cum este o bibliotecă C care prezintă o interfață în stil Visual Basic cu programul gazdă.

  Există și alte instrumente care vă ajută să folosiți interfețe API. Puteți prelua date de ieșire produse de `mysqldump` care descriu structuri de date și le puteți converti într-un fișier antet C. Acest fapt este util la scrierea programelor C care folosesc interfața API MySQL pentru C în vederea accesului la tabel. Dacă folosiți Tcl/Tk, este de asemenea disponibil un generator de ecran de interfață.
- **Pachete care extind caracteristicile sistemului MySQL.** Acestea includ instrumente pentru căutarea de text cu facilități complete și instrumente care vă ajută să reproduceți bazele de date MySQL.
- **Instrumente de conversie a datelor.** Acestea includ programe care facilitează conversia datelor folosite în alte programe în vederea utilizării în MySQL și invers. De exemplu, puteți converti fișierele `.dbf` în tabele MySQL și viceversa, respectiv tabele Access în tabele MySQL și invers.
- **Suport pentru ODBC.** Standardul ODBC (Open Database Connectivity) dezvoltat de Microsoft permite programelor compatibile ODBC să obțină acces la bazele de date MySQL, dacă este disponibil un driver MySQL. De exemplu, puteți instala MyODBC în Windows și apoi puteți folosi programe ODBC sub Windows pentru a obține acces la MySQL. O aplicație comună care implică sistemul MyODBC constă în conectarea la un server MySQL din Microsoft Access. Pentru aceasta, este necesar ca MyODBC să fie configurat sub Windows și trebuie să creați un nume al sursei de

date (DSN) pentru conectarea la MySQL. (Dacă nu ați configurat MyODBC sau nu ați creat un DSN, consultați Anexa A, „Obținerea și instalarea programelor".) După ce ODBC a fost configurat, folosiți următoarea procedură pentru a vă conecta la serverul MySQL din Access:

  1. Porniți programul Access.
  2. Creați o nouă bază de date vidă.
  3. Din meniul File, selectați External Data, apoi Link Tables.
  4. În fereastra care apare, executați clic pe meniul pop-up Files of type și selectați ODBC Databases.
  5. Selectați numele DSN pe care l-ați specificat în panoul de control ODBC pentru a vă conecta la MySQL.
  6. Începeți operația de legare.
  7. Selectați tabelele pe care doriți să le vedeți.
  8. Pentru a deschide un tabel, executați dublu clic pe numele acestuia.
- **Interfețe cu alte programe.** Numeroase instrumente au fost concepute pentru a vă ajuta să încorporați MySQL drept componentă back end pentru alte programe. Există remedii pentru Apache și WU-ftpd care determină aceste programe să execute consemnarea în tabelele MySQL. Puteți folosi MySQL pentru a executa autentificarea utilizatorilor de programe poștale pentru `qmail` și serverul IMAP Cyrus. Autentificarea cu programul `pam` se poate executa folosind de asemenea sistemul MySQL.
- **Instrumente administrative.** Programele care facilitează gestionarea instalării sistemului MySQL se prezintă sub diferite forme. Unele sunt programe în linie de comandă, iar multe altele sunt instrumente GUI. Unii clienți grafici se bazează pe X Windows System, un altul rulează sub Windows, iar alții, scriși în Perl și PHP, sunt bazați pe Web.

ANEXA J

Furnizori de servicii Internet

Mulți oameni au acces complet[1] la Internet și la MySQL prin intermediul unei conexiuni rapide. Aceasta se întâmplă mai ales în medii universitare și în cadrul unor companii suficient de mari pentru a-și avea propriile lor departamente de prelucrare automată a datelor. Dar pentru multe alte persoane această situație nu este valabilă, iar accesul la serviciile on-line este asigurat de un furnizor de servicii Internet (Internet Service Provider - ISP). În afară de conectarea elementară la Internet pentru servicii precum poșta electronică și navigarea prin Web, mulți furnizori oferă și alte servicii, precum accesul la MySQL și un server Web.

Acest capitol reprezintă un ghid pentru selectarea unui furnizor de servicii Internet adecvat cerințelor dumneavoastră. Puteți avea nevoie atât de o modalitate de a vă conecta la Internet, cât și de o gazdă care furnizează sistemul MySQL. Alternativ, este posibil ca MySQL să ruleze deja pe calculatorul dumneavoastră și aveți nevoie de un furnizor la care să vă puteți conecta și care va permite accesul la calculatorul dumneavoastră a cererilor de conexiune sosite prin Internet.

O modalitate de a-i depista pe potențialii furnizori este de a vizita siturile Web pentru a căuta pachetele de care aveți nevoie, cu scopul de a vedea dacă acestea conțin o pagină despre furnizorii care găzduiesc serviciul. De exemplu, situl Web PHP are o pagină de căutare destinată găsirii furnizorilor de servicii Internet care oferă serviciul PHP. Unul dintre criteriile de căutare vă permite să căutați furnizorii care oferă MySQL. Există de asemenea un sit care enumeră furnizorii sistemului MySQL. Puteți vizita aceste situri la următoarele adrese:

```
http://hosts.php.net/search.php3
http://www.wix.com/mysql-hosting/
```

Nu toți furnizorii sunt la fel, deci este important să faceți o analiză comparativă înainte de a alege unul. Alegeți cu atenție; un furnizor de proastă calitate vă poate „lecui" de plăcerea utilizării sistemului MySQL, în timp ce unul bun vă poate ajuta enorm. Veți asuda puțin căutând, dar, dacă nu căutați, puteți face o alegere neinspirată și apoi va trebui oricum să investiți timpul respectiv.

Majoritatea criteriilor prezentate în această anexă trebuie evaluate pe o bază relativă. S-ar putea să nu găsiți un furnizor care să vă satisfacă integral pretențiile, dar trebuie să puteți găsi unul care satisface aceste cerințe mai bine decât alții.

[1] Adică 24 de ore pe zi. – N.T.

Când căutați, încercați să vă păstrați perspectiva. Nu toți furnizorii de servicii Internet sunt „băieți răi", care încearcă să vă jumulească de o sumă de bani cât mai mare și să vă ofere servicii de o valoare cât mai mică. Dar asta nu înseamnă că sunt „băieți buni" cu toții. Este important să vă faceți „temele" așa cum trebuie, astfel încât să găsiți unul dintre furnizorii care își cunosc marfa și care vă pot ajuta să vă atingeți scopurile. În timpul investigațiilor, fiți atent la furnizorul care evită să răspundă la întrebări concrete cu privire la practicile de lucru sau la caracteristicile tehnice. De asemenea, feriți-vă de furnizorii care dau răspunsuri generale atunci când dumneavoastră cereți detalii.

O notă pentru furnizorii de servicii: MySQL reprezintă un serviciu pe care îl puteți oferi clienților dumneavoastră, cu prețul unor costuri financiare mult mai reduse decât ale aproape oricărui alt sistem de baze de date. Desigur, puteți folosi MySQL și intern, pentru propriile dumneavoastră activități (de exemplu, pentru a vă ajuta să păstrați evidența înregistrărilor aferente clienților). Dacă intenționați să instalați sistemul MySQL, citiți capitolul „pe invers". Adică: acolo unde se spune „Întrebați furnizorul dacă așa și pe dincolo", întrebați-vă dacă puteți răspunde sau nu la întrebarea în cauză. Dacă nu puteți, de ce? Acolo unde se pune întrebarea „Asigură furnizorul serviciile cutare sau cutare?", puneți-vă întrebarea dacă oferiți sau nu serviciile respective. Dacă nu, ce este necesar pentru a le oferi?

Primii pași în căutarea unui ISP

Prima etapă în identificarea unui furnizor de servicii este estimarea propriilor dumneavoastră necesități. Înainte de a găsi un furnizor care vă satisface pretențiile, trebuie să știți care sunt cerințele dumneavoastră în ceea ce privește serviciile și lărgimea de bandă. Poate că am afirmat un truism; din păcate însă, unii caută un ISP pornind numai de la ideea vagă că „vor să se conecteze". Un deziderat ca acesta este greu de realizat într-o manieră satisfăcătoare; multe întrebări sunt lăsate fără răspuns, deoarece nici măcar nu au fost puse. Conectat pentru ce? Pentru a realiza ce fel de activități? Care este costul considerat rezonabil?

După ce știți ceea ce vreți, puteți aborda furnizorii de servicii Internet cu un set comun de întrebări și îi veți putea compara mai bine. Dacă nu determinați în prealabil ceea ce doriți să știți, o dată cu fiecare furnizor veți învăța care sunt întrebările ce trebuie puse, veți obține informații incomplete și va fi mai dificil să le comparați.

Ca o componentă a evaluării dumneavoastră, puteți da unele telefoane preliminare pentru a vedea dacă puteți găsi un ISP sau doi care oferă un serviciu de evaluare, pentru a vă ajuta să vă determinați necesitățile. Dacă serviciul este efectiv orientat spre a vă ajuta să vă specificați cerințele (fără a fi doar o altă tehnică de vânzări pentru a vă „vârî pe gât" servicii pe care nu le doriți sau de care nu aveți nevoie), atunci poate că ați găsit un furnizor care are cel puțin un avantaj: un serviciu de relații cu clienții.

În timpul căutărilor dumneavoastră, rețineți că unele instituții care furnizează servicii, precum accesul la MySQL și găzduirea unui sit de Web, nu furnizează un serviciu de conectare prin apel telefonic. Poate că în cazul dumneavoastră soluția cea mai bună constă în alegerea unui furnizor pentru conectarea elementară prin apel telefonic, care vă permite accesul la Internet, respectiv a altui furnizor pentru activități legate de baze de date și Web. (Sau poate că aveți deja un furnizor pentru accesul prin linie telefonică de

care sunteți mulțumit, caz în care trebuie să găsiți un furnizor numai pentru servicii de baze de date și Web.) Secțiunile următoare, intitulate „Lărgime de bandă" și „Servicii", descriu tipurile de probleme rezolvate în modul cel mai adecvat de fiecare tip de ISP.

Lărgime de bandă

Furnizorii oferă diverse opțiuni, de la linii dedicate (mai rapide) la conexiuni prin linie telefonică și modem (mai lente). În general, se face un compromis între cost și viteză: accesul mai rapid costă mai mult. Dacă veți deplasa o mare cantitate de date între calculatorul dumneavoastră și gazda furnizorului, o conexiune prin modem poate fi prea lentă. Totuși, în anumite situații, vă puteți descurca folosind o conexiune mai lentă între calculatorul dumneavoastră și ISP. De exemplu, dacă majoritatea activităților pe care le executați cu gazda furnizorului se datorează clienților care obțin accesul la baza dumneavoastră de date din amonte (prin intermediul unui sit Web, de exemplu), atunci majoritatea traficului dumneavoastră legat de bazele de date va fi orientat dinspre ISP spre exterior.

Servicii

Necesitatea cea mai evidentă pentru orice potențial ISP este aceea de a furniza MySQL, dar și următoarele servicii pot fi necesare sau de dorit:

- **Un cont de e-mail.** Acesta este esențial pentru comunicarea cu personalul tehnic de asistență al furnizorului și îl puteți folosi pentru înscrierea într-una sau mai multe dintre listele de corespondență conexe sistemului MySQL.
- **Acces prin Telnet la un cont de shell.** Acesta vă oferă posibilitatea de a rula utilitare UNIX standard și de a folosi clienții MySQL în linie de comandă precum `mysql`, `mysqldump` și `mysqlimport`. De asemenea, vă veți putea instala propriile dumneavoastră programe.
- **Servicii suplimentare conexe sistemului MySQL.** Acestea pot include găzduirea unui sit Web, PHP și funcționalități CGI.
- **FTP.** Acesta este util pentru transferul fișierelor între calculatorul dumneavoastră și sistemul furnizorului ISP. De exemplu, probabil că aveți unele fișiere de date pe care doriți să le încărcați în baza dumneavoastră de date pentru a o popula inițial sau puteți genera ieșiri de la baza de date pe care doriți să le manipulați pe propriul dumneavoastră calculator. De asemenea, doriți să permiteți altor persoane să descarce fișierele dumneavoastră pe sistemele proprii.
- **Înregistrarea numelor de domenii.** Prin intermediul acestei operații, primiți adrese de e-mail și sit Web sub propriul dumneavoastră nume de domeniu, și nu sub domeniul furnizorului dumneavoastră. Acesta este un serviciu oportun dacă doriți să creați o prezență Web sau FTP sub un nume care să poată fi identificat imediat cu instituția sau cu firma dumneavoastră.

Evaluarea unui ISP – considerații generale

Această secțiune enumeră câteva criterii de ordin general pe care le puteți folosi pentru a evalua potențialii furnizori de servicii Internet. Secțiunea următoare va aborda aspecte specifice sistemului MySQL. Când abordați un nou ISP, puneți-vă întrebările următoare:

- ✓ Cât este de simplă obținerea informațiilor?
- ✓ Care sunt costurile?
- ✓ Care sunt tipurile de programe pentru accesul clienților oferite de ISP?
- ✓ Ce fel de asistență inițială primiți din partea furnizorului?
- ✓ Dispune furnizorul de o conectivitate și de o lărgime de bandă adecvate?
- ✓ Care sunt opțiunile furnizorului pentru conectarea clienților?
- ✓ Ce fel de asistență tehnică oferă furnizorul?
- ✓ Există cote? Dacă da, care sunt acestea?
- ✓ Ce tip de echipamente oferă furnizorul?
- ✓ Cum rezolvă furnizorul problemele legate de confidențialitate și securitate?
- ✓ Ce fel de reputație are furnizorul și de când își desfășoară activitatea în domeniu?
- ✓ Care sunt planurile de dezvoltare ale firmei care furnizează serviciile?

Să discutăm în detaliu despre fiecare din aceste întrebări:

- ✓ **Cât este de simplă obținerea informațiilor?** Când sunați pentru a vă interesa cu privire la serviciile pe care le furnizează un ISP, răspunde acesta la telefon? Dacă aveți probleme în a obține o linie deschisă, poate că este vorba despre o activitate care se desfășoară serile, în garajul cuiva. Dacă vă răspunde un robot telefonic, veți primi un telefon de la furnizor?

  Operatorii serviciului de relații cu clienții cunosc răspunsurile la întrebările dumneavoastră? Dacă nu, pot găsi pe cineva din firmă care să le cunoască? Sau încearcă să vă convingă că problemele pe care le ridicați nu sunt importante (ceva de genul „Sigur că da, putem face asta, n-aveți grijă")? Dacă nu vă pot răspunde la întrebări înainte de a pune mâna pe banii dumneavoastră, cum vă vor trata după aceea?

  Puteți înțelege termenii folosiți de personalul furnizorului atunci când discută cu dumneavoastră? Dacă singurul limbaj pe care îl vorbesc este jargonul de marketing, atunci ați luat legătura cu o firmă care ține mai mult să-și vândă serviciile și care nu prea dispune de cunoștințe tehnice solide. Pe de altă parte, dacă vă sufocă de atâta jargon, atunci aveți de-a face cu o adunătură de „expertoizi" care își cunosc bine domeniul, dar care sunt incapabili să comunice adecvat cu oamenii obișnuiți.

  Vizitați situl Web al furnizorului, dacă aveți acces la un browser. Situl este clar, conține informații utile și este ușor de înțeles? Dacă furnizorul nu este în stare să încropească o pagină de bază performantă, atunci care va fi următorul său „rateu"?
- ✓ **Care sunt costurile?** Acestea pot varia considerabil, deoarece depind de un număr foarte mare de factori. Există o taxă inițială de abonament sau configurare în afară de taxa lunară normală? Taxa lunară este fixă sau variază? Ce include taxa? Aveți timp de

conectare nelimitat sau există o limită orară? Dacă depăşiţi limita, ce se întâmplă? Furnizorul refuză conectarea (ceea ce nu este prea convenabil, dar nu vă costă nimic în plus), sau există o taxă de depăşire dincolo de taxa dumneavoastră lunară normală? Dacă există un tarif, este inclus şi spaţiul pe disc?

Pentru conexiunile prin apel telefonic, este de presupus că furnizorul are un număr local pe care îl puteţi folosi în zona dumneavoastră de apelare. Dispune furnizorul de un număr de telefon special pe care îl puteţi folosi atunci când nu vă aflaţi în zona respectivă? Există o taxă suplimentară pentru utilizarea acestui număr? Dacă locuiţi într-o zonă rurală, probabil că opţiunile dumneavoastră sunt mai limitate şi cel mai important este să vă asiguraţi că accesul la furnizor nu implică un apel cu taxă.

Deseori, furnizorii includ în taxa de bază o cantitate modestă de spaţiu pe disc (câţiva megaocteţi), cu opţiunea de a cumpăra o porţiune mai mare, dacă este necesar. Obţineţi informaţii exacte referitoare la taxele pentru utilizarea discului, dacă vă aşteptaţi la baze de date de mari dimensiuni sau dacă intenţionaţi să găzduiţi un sit Web (mai ales dacă va conţine numeroase elemente grafice).

Dacă vă creaţi o prezenţă Web sau FTP pe gazda furnizorului, care sunt taxele percepute de acesta pentru volumul fişierelor descărcate atunci când utilizatorii obţin accesul la paginile dumneavoastră de Web sau când descarcă fişiere? Unii furnizori impun o cotă şi, dacă depăşiţi limita, sunteţi blocat până la începutul perioadei următoare. Alţii pot include în nota dumneavoastră de plată o taxă variabilă, bazată pe volumul fişierelor descărcate. Iar alţii vă pot schimba categoria de clasificare în „client comercial“ (probabil la o taxă mai mare). Aflaţi care sunt opţiunile aflate la dispoziţia dumneavoastră.

Asistenţa tehnică este gratuită sau există o taxă? Dacă este gratuită, este numai pentru o perioadă limitată de timp, de exemplu pentru primele 30 de zile? Dacă există o taxă, aceasta este fixă sau există o taxă per incident? Se poate achita o taxă suplimentară pentru a beneficia de asistenţă în condiţii speciale?

Dacă vă este prezentat un plan de achitare în avans a taxei, evaluaţi-l ţinând cont de istoricul şi de reputaţia companiei. Opţiunile de plată în avans sunt uneori folosite de firmele aflate la început, ca mijloc de acumulare a unui capital de operare, dar asemenea furnizori au tendinţa de a dispărea de pe piaţă, caz în care taxele dumneavoastră plătite în avans sunt irecuperabile.

Trebuie să semnaţi un contract pe termen lung (de un an sau mai mult)? Dacă nu sunteţi sigur de posibilităţile unui furnizor, doriţi să aveţi posibilitatea de a renunţa la serviciile acestuia după o lună-două. E uşor să închei un abonament; cât de uşor este să anulaţi abonamentul, dacă serviciile firmei nu vă satisfac? Vă puteţi recupera porţiunea nefolosită a contractului dumneavoastră?

Poate vă puteţi procura un cont de testare gratuit sau la un preţ redus, care vă va permite să estimaţi competenţa tehnică şi serviciile de asistenţă ale furnizorului. Reţineţi că, în mod caracteristic, un asemenea cont va include limitări ale activităţilor dumneavoastră. De exemplu, restricţiile privind spaţiul pe disc vor limita dimensiunea oricărei baze de date de test pe care o puteţi crea.

- ✓ **Care sunt tipurile de programe pentru accesul clienţilor oferite de ISP?** Furnizorul acceptă platforma dumneavoastră? Unii furnizori pot specifica faptul că acceptă numai anumite sisteme, cum este Windows, ceea ce nu vă este de folos dacă rulaţi UNIX sau Mac OS.[2]

  Oferă furnizorul un client de e-mail şi un browser Web, dacă aveţi nevoie de aceste servicii? Trebuie să le folosiţi pe cele oferite de furnizor sau puteţi folosi şi altele, dacă preferaţi?
- ✓ **Ce fel de asistenţă iniţială primiţi din partea furnizorului?** Oferă furnizorul asistenţă în ceea ce priveşte conectarea şi punerea în funcţiune a contului dumneavoastră? Există o taxă pentru acest serviciu (ceea ce este posibil dacă aveţi cerinţe complexe)?
- ✓ **Dispune furnizorul de o conectivitate şi de o lărgime de bandă adecvate?** Cât de aproape este furnizorul de coloana vertebrală a Internetului? Furnizorul este conectat direct sau se foloseşte de un alt furnizor pentru a ajunge acolo? Furnizorul dispune de conexiuni redundante în cazul apariţiei unei pene de conexiune?

  Cât de mare este „conducta“ dintre ISP şi coloana vertebrală? Tipul de conexiune determină *lărgimea de bandă* a acesteia, adică volumul maxim de trafic pe care îl poate manipula furnizorul între coloana vertebrală şi propriile sale sisteme. Liniile T-3 şi T-1 sunt trunchiuri de lărgime de bandă mare, care asigură conexiuni rapide, dar, în cazul în care propriile conexiuni de ieşire ale furnizorului se realizează prin modeme, va apărea o strangulare şi vitezele de transmisie vor avea de suferit.

  Interesaţi-vă cu privire la traficul care trece prin conexiunea furnizorului – care este procentul de utilizare al capacităţii acestuia? Dacă împărţiţi cu mulţi alţi utilizatori activi o conexiune la coloana vertebrală, lărgimea de bandă disponibilă pentru dumneavoastră scade, ceea ce nu vă va încetini numai pe dumneavoastră; o legătură saturată reduce viteza de acces a persoanelor care se conectează din amonte la baza de date sau la situl dumneavoastră de Web, dacă aveţi unul.
- ✓ **Care sunt opţiunile furnizorului pentru conectarea clienţilor?** Dumneavoastră puteţi dispune de un modem la 56Kbps, dar şi furnizorul trebuie să-l accepte; în caz contrar, veţi fi limitat la viteze de conexiune mai reduse. Oferă furnizorul opţiuni de viteză mai mare, precum ISDN sau DSL? Dacă vă aşteptaţi să manipulaţi cantităţi mari de trafic între situl dumneavoastră şi furnizor, este de dorit chiar să configuraţi o linie dedicată. Vă poate ajuta furnizorul în acest sens?

Nu cereţi prea mult de la un ISP

Nu aveţi cum să beneficiaţi de acces nelimitat, spaţiu pe disc nelimitat şi asistenţă tehnică nelimitată pentru 20 de dolari pe lună. Sunteţi dispus să plătiţi pentru un serviciu şi o asistenţă de calitate? Doriţi să primiţi ceva în schimbul banilor dumneavoastră, dar şi furnizorii se aşteaptă la compensaţii rezonabile pentru serviciile pe care le oferă.

[2] Dacă vă conectaţi prin intermediul protocoalelor PPP (protocol punct la punct) sau SLIP (IP prin linie serială), probabil că vă puteţi conecta de la practic orice platformă. – N.A.

Dacă vă veți conecta cu ajutorul modemului, nu doriți să primiți un semnal „ocupat" atunci când încercați să ajungeți la furnizorul de servicii. Vă puteți interesa la diverșii furnizori care este raportul dintre clienți și modeme, dar aceasta depinde de activitatea clienților. Clienții rezidențiali, care se conectează pentru scurtă vreme, nu ocupă rezerva de modeme în aceeași măsură ca și clienții comerciali, care ocupă toată ziua o conexiune pentru a-și dirija afacerile. Cel mai bine este să întrebați cât de frecvent veți primi un semnal de linie ocupată, mai ales la ora din zi la care vă veți conecta.

Puteți face următorul test înainte de a vă abona: formați numărul de telefon al furnizorului de la un telefon obișnuit și ascultați scârțâiturile modemului aflat la celălalt capăt al firului. Efectuați acest test de mai multe ori, în mai multe zile. Cât de des telefonul sună ocupat sau nu primiți nici un răspuns? Comparați disponibilitatea modemului cu activitățile dumneavoastră. Dacă sunteți o firmă, doriți să fiți sigur că vă puteți conecta ori de câte ori doriți. Pentru scopuri personale, poate că semnalul ocazional de linie ocupată nu vă deranjează.

- ✓ **Ce fel de asistență tehnică oferă furnizorul?** Asistența tehnică este disponibilă permanent sau numai în timpul orelor normale de serviciu? Poate că intenționați să vă conectați numai în timpul zilei; în caz contrar, se va găsi cineva care să vă răspundă la întrebare? Personalul de asistență tehnică dispune și de telefon, și de e-mail? Aveți nevoie de un telefon, deoarece nu puteți trimite mesaje e-mail dacă nu vă puteți conecta. Dar doriți să beneficiați și de e-mail, pentru a putea trimite personalului de asistență informații tehnice sau date de ieșire ale programului, pentru a evita încercarea de a le descrie la telefon. Vi s-a promis un răspuns într-un anumit interval de timp? (Este posibil ca problema să nu se rezolve în intervalul respectiv, dar cererile trebuie măcar confirmate rapid, pentru ca să știți că se lucrează la rezolvarea lor.)

  Există asistență on-line? Este clară și la obiect sau derutantă și vagă? O puteți parcurge cu ușurință, pentru a găsi informațiile de care aveți nevoie? Se pot executa căutări?

  Puteți evalua personal disponibilitatea asistenței tehnice în același mod în care testați numerele de telefon pentru acces – dați-le un telefon. Poate că vi se pare ciudat să faceți asta înainte de a avea un cont la furnizorul respectiv, dar poate fi instructiv să determinați dacă sunteți pus sau nu pe „apel în așteptare" și pentru cât timp. De asemenea, întrebați care este numărul celor care lucrează în departamentul respectiv. Mai bine interesați-vă care este raportul client-tehnician.

- ✓ **Există cote? Dacă da, care sunt acestea?** Există cote pentru spațiul pe disc sau pentru timpul de prelucrare? Există restricții de timp pentru scripturile executate de serverul Web? Nu este ilogic ca furnizorul să impună unele limite pentru activitățile clientului, pentru a preveni monopolizarea resurselor partajate de alți clienți, dar doriți să știți care sunt aceste limite.

- ✓ **Ce tip de echipamente oferă furnizorul?** Se pot rula servicii Internet elementare pe un PC vechi de clasă 386, dar această soluție nu este adecvată pentru numere mari de utilizatori, pentru a nu mai aminti nimic de încercarea de a rula servicii care necesită o putere mai mare de prelucrare a datelor, precum MySQL. Acesta nu acaparează resursele sistemului, așa cum procedează sistemele de baze de date de mari dimensiuni, dar are totuși nevoie de un ISP care să ruleze echipamente cu o oarecare cantitate de... mușchi. Care sunt categoriile de încărcări pe care sistemul le poate manipula fără probleme?

  Unde se află serverul? În garajul sau în pivnița cuiva? Un furnizor situat într-un district comercial poate convinge mai ușor compania de telefoane să-i aloce linii de trunchi suplimentare, pe măsura creșterii cerințelor.

  Cum rezolvă furnizorul problema defecțiunii echipamentelor? Există un plan de refacere, sau sunteți pur și simplu îmbrâncit afară din rețea până la repararea sau înlocuirea echipamentului? Care este procentajul real de funcționare al furnizorului în ultimele luni? Cine este responsabil cu incidentele care cauzează căderea sistemului? Se execută copii de siguranță ale sistemului de fișiere, și dacă da, cât de frecvent? Sau, după părerea furnizorului, aceasta este responsabilitatea dumneavoastră?

  Există timpi de nefuncționare planificați (pentru operațiile planificate de întreținere sau executarea copiilor de siguranță, de exemplu)? Îi informează furnizorul pe clienți cu privire la momentul de început al acestor perioade? Dacă rulați un server Web, probabil preferați ca acesta să fie disponibil 24 de ore pe zi.

- ✓ **Cum rezolvă furnizorul problemele legate de confidențialitate și securitate?** Are furnizorul o politică legată de confidențialitatea fișierelor dumneavoastră? Ce măsuri ia acesta pentru a preveni compromiterea contului dumneavoastră?

- ✓ **Ce fel de reputație are furnizorul și de când își desfășoară activitatea în domeniu?** Care este reputația furnizorului? Ce alți clienți mai are și cât sunt de satisfăcuți aceștia? Informațiile verbale pot fi utile; întrebați-vă cunoștințele, pentru a afla dacă au auzit de potențialii furnizori. Puteți cere furnizorului referințe și poate fi util să solicitați păreri în lista de corespondență MySQL. Probabil că un furnizor vă va trimite la persoane care sunt satisfăcute de serviciile acestuia. În lista de corespondență, probabil că aveți auzi păreri din ambele tabere. Abonații de acolo sunt foarte doritori să relateze despre calitatea bună sau proastă a experienței lor cu un furnizor.

  De cât timp activează furnizorul respectiv? Industria furnizorilor de servicii Internet înregistrează un număr fenomenal de mare de firme falimentare sau de scurtă durată, deoarece a porni este mai ușor – a te menține este mai dificil. Un furnizor cu o oarecare longevitate are mai multe șanse de a exista și în viitorul mai îndepărtat și poate fi capabil de a vă ajuta pe termen lung.

- ✓ **Care sunt planurile de dezvoltare ale firmei?** Care este mărimea furnizorului? Probabil că nu doriți să folosiți o firmă mică-micuță, dar chiar nici cele mai mari servicii nu sunt în mod obligatoriu mai bune. Companiile mai mici pot fi mai apropiate de necesitățile propriilor clienți.

  Care este ritmul de dezvoltare a firmei? Care este numărul de clienți de care dispun în prezent? Care era acest număr acum o lună? Dar acum un an? Puteți dori să colaborați cu o companie prosperă, dar dezvoltarea exercită presiuni asupra resurselor privind lărgimea de bandă, accesul clienților și asistența tehnică. Care este politica firmei de rezolvare a problemelor legate de creșterea încărcării?

Evaluarea unui ISP – considerații specifice sistemului MySQL

Secțiunea anterioară v-a oferit o modalitate de evaluare a caracteristicilor generale ale unui ISP. Această secțiune discută despre criteriile pentru evaluarea serviciilor MySQL ale unui ISP. Când vedeți scris „MySQL", interpretați această mențiune ca incluzând toate celelalte servicii conexe de care aveți nevoie, precum Apache, PHP și DBI:

- ✓ Care sunt serviciile conexe sistemului MySQL pe care le furnizează un ISP?
- ✓ Care este versiunea sistemului MySQL instalată?
- ✓ Care este politica furnizorului în ceea ce privește actualizarea sistemului MySQL?
- ✓ Sistemul de operare al furnizorului prezintă probleme cunoscute în ceea ce privește sistemul MySQL?
- ✓ Vă puteți instala propriile dumneavoastră programe?
- ✓ Confidențialitatea și securitatea bazei dumneavoastră de date reprezintă o preocupare pentru furnizor?
- ✓ Vă ajută personalul furnizorului la începutul activității dumneavoastră?
- ✓ Care este categoria de asistență tehnică specifică sistemului MySQL asigurată de furnizor?
- ✓ Câți dintre clienții furnizorului folosesc MySQL?

Să discutăm în detaliu despre fiecare dintre aceste întrebări.

- ✓ **Care sunt serviciile conexe sistemului MySQL pe care le furnizează un ISP?** Furnizorul trebuie să aibă deja instalat sistemul MySQL și toate celelalte pachete conexe de care aveți nevoie. Excepția poate fi aceea că aveți deja un furnizor de care sunteți mulțumit din alte puncte de vedere, dar care nu dispune de MySQL. În acest caz, vă puteți interesa la furnizor dacă acesta dorește să-l instaleze pentru dumneavoastră. Va trebui să folosiți un fel de abordare de afaceri pentru aceasta, și va trebui probabil să fiți de acord că o asemenea instalare se va face acceptând realitatea că MySQL nu este un serviciu care beneficiază de nivelul de asistență aferent serviciului de e-mail sau găzduirii de pagini Web.

  Vizitați situl Web al furnizorului. Conține acesta informații (mai mult sau mai puțin detaliate) despre serviciile MySQL oferite? Există o înțelegere de natură tehnică a sistemului sau situl abundă de jargonul folosit în marketing? Situl este ușor de parcurs? Conține acesta o zonă cu răspunsuri la întrebări puse frecvent de clienți cu privire la utilizarea serviciilor proprii, astfel încât să puteți căuta singur informațiile?
- ✓ **Care este versiunea sistemului MySQL instalată?** Este o versiune recentă sau o variantă veche de-a binelea, de care nu s-a mai atins nimeni de la instalarea acesteia? Examinați anexa cu notele de trecere de la o versiune la alta din manualul de referință MySQL, pentru a determina caracteristicile adăugate după versiunea rulată de furnizor. Aveți nevoie de aceste facilități?
- ✓ **Care este politica furnizorului în ceea ce privește actualizarea sistemului MySQL?** Furnizorul trebuie să creeze un echilibru între nevoia de stabilitate și de comportare cunoscută a aplicațiilor MySQL existente cu dorința clienților de a beneficia de noile caracteristici ale variantelor recente. Aceasta nu este în mod obligatoriu o problemă ușor de rezolvat, deci nu tratați preocupările furnizorului vizavi de aceasta ca pe ceva banal. Dar doriți să știți dacă există măcar o posibilitate de trecere la o versiune mai recentă, eventual pe o gazdă separată, special destinată testelor. Dacă vă oferiți să serviți drept cobai pentru variantele mai recente, veți putea ajunge la o soluție de cooperare, deoarece veți asigura furnizorului dumneavoastră un valoros serviciu de testare.
- ✓ **Sistemul de operare al furnizorului prezintă probleme cunoscute în ceea ce privește sistemul MySQL?** Examinați capitolul de instalare din manualul de referință MySQL, pentru a vedea dacă există limitări care vă pot afecta. Deseori, există soluții ocolitoare pentru aceste probleme.
- ✓ **Vă puteți instala propriile dumneavoastră programe?** În caz afirmativ, vi se permite să vă extindeți funcționalitățile propriului dumneavoastră sistem MySQL cu programe create de terțe părți sau concepute chiar de dumneavoastră.
- ✓ **Confidențialitatea și securitatea bazei dumneavoastră de date reprezintă o preocupare pentru furnizor?** În afară de măsurile generale luate de furnizor pentru protecția datelor personale ale clienților, furnizorul ia măsuri pentru protecția datelor MySQL în particular? Aveți propriul dumneavoastră server MySQL? Aceasta ar fi un avantaj din punctul de vedere al clientului, deoarece puteți controla persoanele care au acces la conținutul bazei dumneavoastră de date. În ceea ce privește furnizorul, această opțiune incumbă o cantitate de muncă suplimentară.
- ✓ **Vă ajută personalul furnizorului la începutul activității dumneavoastră?** Nu vă așteptați ca furnizorul să vă învețe SQL și să vă arate cum să vă scrieți interogările, dar trebuie să vă puteți conecta la serverul MySQL. Vi se spune ce anume trebuie să știți pentru a stabili o conexiune? Primiți un script demonstrativ pe care îl puteți rula pentru a verifica dacă aveți acces la baza dumneavoastră de date?
- ✓ **Care este categoria de asistență tehnică specifică sistemului MySQL asigurată de furnizor?** Din când în când, oamenii trimit mesaje în lista de corespondență MySQL, care conțin problemele avute cu furnizorul și la încercarea de a determina sistemul MySQL să ruleze în mod corespunzător. Am observat că, în mod destul de frecvent, furnizorul indică clientul drept sursă a problemei. Ca în multe alte controverse, este posibil ca în multe cazuri să existe greșeli de ambele părți, dar, din descrierile situațiilor apărute, pare clar că în multe situații furnizorul are prea puține cunoștințe privind modul de manipulare a sistemului MySQL și rezolvă problemele „pe ghicite". Din acest motiv, este important să determinați în prealabil dacă un furnizor potențial dispune sau nu de un personal administrativ sau tehnic care să aibă cunoștințe decente de MySQL. Întrebați-i pe membrii acestui personal care este nivelul lor de experiență tehnică. Interesați-vă dacă dispun de un contract de asistență pentru MySQL. În caz afirmativ, înseamnă că tratează sistemul MySQL cu toată seriozitatea și că sunt pregătiți să vă ajute la remedierea problemelor care pot apărea.

  De exemplu, are furnizorul experiență în manipularea fiecărui pachet de care aveți nevoie, astfel încât să poată oferi o asistență efectivă, sau se limitează să vă trimită la lista de corespondență? Furnizorul trebuie să cunoască materialul prezentat în Partea a III-a a acestei cărți, intitulată „Administrarea sistemului MySQL".

Feriți-vă de un administrator absent

Fiți atent la furnizorii care instalează sistemul MySQL doar ca să-l poată adăuga pe lista lor de servicii și să atragă mai mulți clienți. Nu doriți să aveți de-a face cu administratorul care încearcă să se fofileze cu o cantitate minimă de cunoștințe legate de îndatoririle sale administrative.

Va trebui să comparați asistența tehnică oferită cu nivelul de control asupra resurselor dumneavoastră pe care vi-l oferă furnizorul. Dacă doriți un control complet asupra serverului dumneavoastră MySQL, iar furnizorul vi-l oferă, este logic din partea lui să presupună că sunteți capabil de a administra serverul. În acest caz, doriți ca furnizorul să vă permită să asigurați un script care va porni serverul la încărcarea sistemului de operare al calculatorului, astfel încât să nu fiți obligat să porniți manual serverul la fiecare repornire a calculatorului.

✓ **Câți dintre clienții furnizorului folosesc MySQL?** Dacă sistemul MySQL este folosit în mod activ de mai mulți clienți, sunt șanse mai mari ca furnizorul să asigure o asistență de calitate decât dacă MySQL este folosit numai de unul sau doi clienți.

Index

Simboluri

A

D

E

Q

R

S-Ș

T

U

V

W-Z